

KDN-K3 系列可编程控制器

用户手册



北京凯迪恩自动化技术有限公司
Beijing KDN Automation Co.,Ltd

KDN-K3 系列小型一体化 PLC

用户手册

前　　言

北京凯迪恩自动化技术有限公司是致力于国产高品质 PLC 开发、营销、生产、服务的高科技企业。公司的创业者凭借在自动化领域十几年的实践经验和对 PLC 产品的深刻理解，利用我们所掌握的 PLC 软/硬件核心技术，开发与国际同步的高品质 PLC，以 100%自有知识产权、优良的品质、体贴到位的服务，创建适合国内用户的小型一体化 PLC 民族品牌，打破进口品牌垄断中国市场的局面，为民族自动化产业贡献一份力量。

KDN-K3 系列 PLC 自推向市场以来，已经应用到环保机械、木工机械、纺织机械、建材机械、食品机械、中央空调以及单一过程控制装置，以其质优价廉得到用户的一致认可。

KDN-K3 系列 PLC 在产品测试及试用的过程中，得到广大热心支持民族产业的各界朋友的帮助和鼓励，在此表示衷心感谢，我们坚信凯迪恩 PLC 会得到广大用户的认可，凯迪恩品牌也会一天天成长起来。

为了方便用户使用 KDN-K3 系列 PLC，我们编写了此手册，对软件、硬件进行了系统介绍。我们力求手册内容详尽，对一些 PLC 及 IEC61131-3 的基本概念也进行了介绍，以照顾广大初学 PLC 的用户。

由于时间仓促，书中难免有疏漏不足之处，敬请广大用户指正，我们将不胜感激。

若您需要更进一步了解一些功能的用法及获得更多的技术支持请通过以下方式与我们联系：

地　　址：北京市海淀区成府路 45 号海升写字楼 C 座 308 – 312 室

邮　　编：100083

电　　话：010-62551582、62639677 、62633902/03

传　　真：010-62639677-810

邮箱地址：postmaster@kdnautomation.com

网　　址：http://www.kdnautomation.com

网络实名：“凯迪恩 PLC”

目 录

第一部分

通 用 说 明

第一章　系统通用说明

本章简要描述了KDN-K3系列小型一体化可编程控制器的背景信息,并着重解释了与KDN-K3有关的名词术语，从而有助于用户在后续的阅读中对于本手册内容的理解和掌握。本章的主要内容为：产品应用范围、名词术语解释、命名规则说明等。

1.1 产品应用范围

按照公认的 PLC 分类规则，KDN-K3 系列 PLC 系列属于小型一体化 PLC，因此适用于工厂自动化领域中的机器控制和小规模的过程控制。KDN-K3 系列 PLC 能够满足如下应用领域的需求（但不限于这些应用）：

包装机械	纺织机械	建材机械	食品机械
塑料机械	数控机床	印刷机械	中央空调
环保设备	单一过程控制装置		

1.2 本文常用名称

1.2.1 名词术语

- 小型一体化可编程控制器

按国际通用分类原则，小型 PLC 一般指实际控制点数小于 128 点（并非设计控制点数）的 PLC，此类 PLC 通常采用一体化结构，即在 CPU 模块上集成一定数量的 I/O 以及输出电源、高速输入/输出等其他附件。

- CPU 本体

即 CPU 模块，是控制系统的核心。用户编写的应用程序经编程软件下载后存储于 CPU 模块的内部存储器中，在 CPU 运行软件的调度下执行控制功能，运行软件同时还负责诊断各模块的工作状态和用户程序中的语法错误。

▪ 扩展模块

用于扩展 CPU 本体功能的模块，分为扩展 I/O 模块（增加系统的输入/输出通道）和扩展功能模块（扩展 CPU 功能）。

▪ 扩展总线

连接 CPU 模块和扩展模块的数据和信号通道，物理介质采用了 16 芯扁平电缆。在扩展总线中集成有数据总线、地址总线和扩展模块工作电源。

▪ EasyProg

KDN-K3 系列 PLC 的上位编程软件，符合 IEC61131-3 标准，目前采用支持 LD 和 IL 两种标准语言。利用 EasyProg 可以实现编程、调试等各种功能。

▪ CPU 运行软件

又称 CPU 板级固件（firmware），存储于 CPU 本体的 Flash 存储器中，一旦 CPU 模块上电即开始运行，管理、调度 CPU 的所有任务。用户编写的应用程序是在 CPU 运行软件的调度之下执行的。运行软件同时还负责诊断各模块的工作状态和用户程序中的语法错误。

▪ 应用程序

又称用户程序、用户工程，是用户编写的完成特定控制功能的程序。应用程序下载后存储于 CPU 模块的 FRAM（铁电存储器）中，CPU 上电运行时被读入 RAM 中执行。

▪ 程序扫描

在 CPU 中主程序是被循环连续执行的，CPU 连续执行用户主程序的过程称为扫描。

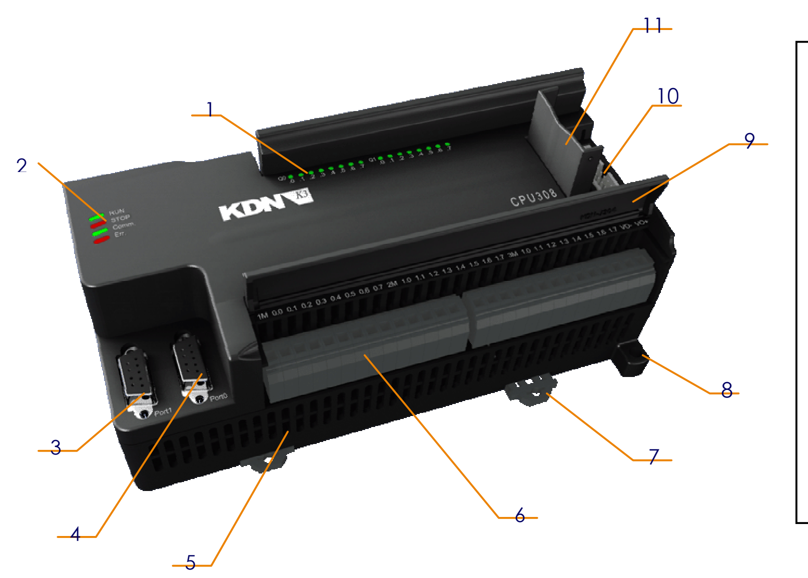
▪ 主程序

应用程序的执行入口，所谓的程序扫描就是 CPU 对主程序的扫描，应用程序中只能有唯一的主程序，但可以从主程序中调用多个子程序。

▪ I/O 映像区

物理 I/O 点的状态数据在 CPU 中的存储区域，分为输入映像区和输出映像区。为保证 CPU 在一次程序循环中数据的一致性及提高程序执行速度，在每次扫描中 CPU 首先将输入点的状态读入到输入映像区，应用程序执行过程中只对 I/O 映像区进行访问，在扫描结束时再将输出映像区中的数据发送到输出通道。

1.2.2 产品的结构件名称



1、 I/O 状态指示灯
2、 CPU 状态指示灯
3、 编程口（RS232）
4、 通讯接口（RS485）
5、 散热孔
6、 接线端子
7、 35mm 导轨卡接滑块
8、 M4 螺栓安装孔
9、 端子盖板

图 1-1　模块各结构件说明

1.3 产品命名规则

1.3.1 产品名称说明

KDN-K3 PLC 模块的“产品名称”用于表示该产品的主要功能和用途。“产品名称”不是唯一的，而是一类产品的总称。“产品名称”按以下原则确定：

产品名称： ***模块类型 ＋ 3 ＋子类型 ＋ 流水号***

- *模块类型*：用如下英文字母来表示。

  CPU　主控模块

  PM　扩展 I/O 模块

  FM　扩展功能模块

  SW　软件

  AS　附件

- *3*　：代表 KDN-K3 系列小型一体化 PLC。

- *子 类 型*：采用用一位数字（0～9）表示模块的子类型。

  0　CPU 模块

  1　保留

  2　开关量模块

  3　模拟量模块

  4　通讯模块

  5　特殊功能模块

  6　系统软件

  7　附件

  8　保留

9　保留

- *流 水 号*: 用一位数字（0～9）来表示某一子类型中的序号。各子类型中流水号含义如下。

▪ CPU 模块

4　表示 CPU 本体自带 14 路 I/O 通道；

6　表示 CPU 本体自带 24 路 I/O 通道；

8　表示 CPU 本体自带 40 路 I/O 通道；

其余流水号保留。

▪ 开关量模块

1　表示是开关量输入模块；

2　表示是开关量输出模块；

3　表示是开关量输入/输出混合模块；

其余流水号保留。

▪ 模拟量模块

1　表示是模拟量输入模块；

2　表示是模拟量输出模块；

3　表示是模拟量输入/输出混合模块；

其余流水号保留。

▪ 通讯模块

0　表示是 RS485 通讯模块；

2　表示是 Profibus-DP 协议从站接口模块；

3　表示是 DeviceNet 协议接口模块；

4　表示是 Ethernet 接口模块；

6　表示是 Modem 接口模块；

其余流水号保留。

▪ 特殊模块

0　表示是运动控制模块；

其余流水号保留。

- **系统软件**

0　表示是编程软件；

其余流水号保留。

- **软件及附件**

0　表示是编程电缆；

1　表示是 Profibus-DP 连接器；

2　表示是 Profibus-DP 中继器；

其余流水号保留。

按照以上原则，*CPU306* 表示带 24 点 I/O 的 CPU 模块；*PM321* 表示扩展开关量输入模块；*AS360* 表示 EasyProg 编程软件，等等…

1.3.2 订货号说明

与“产品名称”不同，每一种产品（模块）都有唯一的“订货号”，用户在订货时只要清晰地告诉我们所需产品的订货号即可。产品“订货号”按以下原则确定：

订货号：　*KDN-K ＋ 模块代号 ＋ 特征代码*

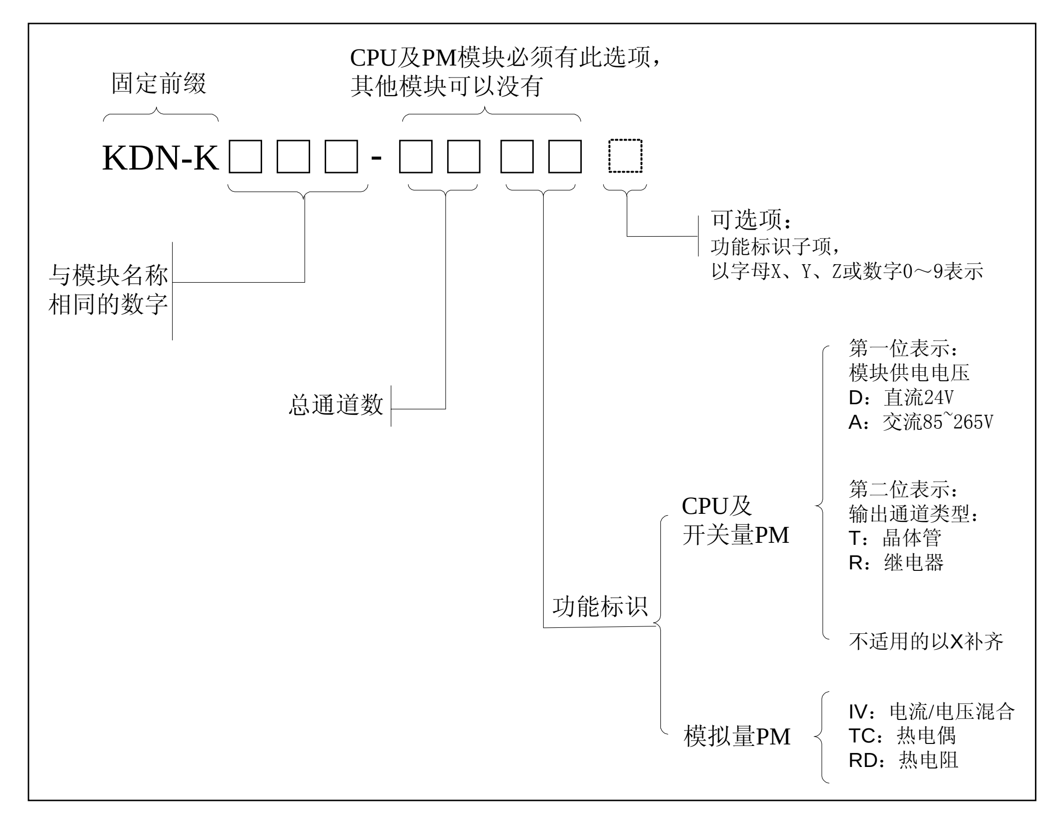


图 1-2　“产品订货号”的说明

按照以上的原则，*KDN-K306-24DT* 表示 DC24V 供电，带 24 点 I/O（其中输出通道类型为晶体管）的 CPU 模块；*KDN-K321-08DX* 表示 8 通道 DC24V 开关量输入扩展模块。

1.4 KDN-K3 系列 PLC 产品列表

类型	名 称	订货号	性能描述
CPU 模块	CPU304	KDN-K304-14DT	DC24V 供电，DI 8×DC24V，DO 6×DC24V，DO 是晶体管输出，每通道最大输出电流 0.75A
		KDN-K304-14AR	AC85~265V 供电，DI 8×DC24V，DO 6×继电器，DO 每通道最大输出电流 3A
	CPU306	KDN-K306-24DT	DC24V 供电，DI 14×DC24V，DO 10×DC24V，DO 是晶体管输出，每通道最大输出电流 0.75A
		KDN-K306-24DR	DC24V 供电，DI 14×DC24V，DO 10×继电器，DO 每通道最大输出电流 3A
		KDN-K306-24AR	AC85~265V 供电，DI 14×DC24V，DO 10×继电器，DO 每通道最大输出电流 3A
	CPU308	KDN-K308-40DT	DC24V 供电，DI 24×DC24V，DO 16×DC24V，DO 是晶体管输出，每通道最大输出电流 0.75A
		KDN-K308-40AR	AC85~265V 供电， DI 24×DC24V，DO 16×继电器，DO 每通道最大输出电流 3A
扩展 I/O 模块	PM321	KDN-K321-08DX*	DI 8×DC24V
		KDN-K321-08AX	DI 8×AC220V
		KDN-K321-16DX	DI 16×DC24V
	PM322	KDN-K322-08DT*	8×DC24V，晶体管输出，每通道最大输出电流 0.75A
		KDN-K322-16DT	16×DC24V，晶体管输出，每通道最大输出电流 0.75A
		KDN-K322-08XR*	8×继电器，每通道最大输出电流 3A
		KDN-K322-16XR	16×继电器，每通道最大输出电流 3A
	PM323	KDN-K323-08DTX*	DIO 8×DC24V，晶体管型 DI、DO 复用，每通道最大输出电流 0.75A
		KDN-K323-08DT	DI 4×DC24V，DO 4×DC24V，晶体管型，每通道最大输出电流 0.75A
		KDN-K323-08DR	DI 4×DC24V，DO 4×继电器，每通道最大输出电流 3A
		KDN-K323-16DT	DI 8×DC24V，DO 8×DC24V，晶体管型，每通道最大输出电流 0.75A
		KDN-K323-16DR	DI 8×DC24V，DO 8×继电器，每通道最大输出电流 3A

	PM331	KDN-K331-04IV	4 通道模拟量输入，4-20mA/1~5V/0~20mA/±10V 可选
		KDN-K331-02IV	2 通道模拟量输入，4-20mA/1~5V/0~20mA/±10V 可选
		KDN-K331-04TC	4 通道热电偶输入
		KDN-K331-02RD	2 通道热电阻输入（2，3 线可选）
	PM332	KDN-K332-02IV	2 通道模拟量输出，4-20mA/1~5V/0~20mA/±10V 可选
	PM333	KDN-K333-05IV	4 通道模拟量输入，4-20mA/1~5V/0~20mA/±10V 可选； 1 通道模拟量输出，4-20mA/1~5V/0~20mA/±10V 可选
扩展功能模块	FM340	KDN-K340	Modbus 协议/无协议 RS485 通讯模块， 最大通讯速率 187.5Kbps，可支持 32 个站通讯
	FM342	KDN-K342	Profibus-DP 协议从站接口模块， 速率 12Mbps，从站地址可设置，最大输入/输出字节 128/128
	FM343	KDN-K343	DeviceNet 协议接口模块
	FM344	KDN-K344	Ethernet 接口模块，10/100Mbps 自适应，RJ45 接口
	FM346	KDN-K346	调制解调器模块
	FM350	KDN-K350	定位/运动控制模块，脉冲输出最大 200KHz，
软件及附件	SW360	KDN-K360-Vx.x	EasyProg，符合 IEC61131-3 标准的编程软件（x.x 代表版本号）
	AS370	KDN-K370-020	2 米编程电缆
		KDN-K370-050	5 米编程电缆
		KDN-K370-100	10 米编程电缆
	AS371	KDN-K371	Profibus-DP 连接器
	AS372	KDN-K372	Profibus-DP 中继器
	AS373	KDN-K373-XXX	加长总线扩展电缆
	AS374	KDN-K374	总线终端器

表 1-1　产品列表

第二部分

硬 件 手 册

第一章 KDN-K3 系列 PLC 应用

本章对 KDN-K3 系列小型一体化可编程控制器的体系结构、扩展连接、网络接口等方面进行了概要性描述，这些描述有助于用户了解和掌握 KDN-K3 的基础知识，从而在应用中正确地使用 KDN-K3 系列 PLC。

1.1　KDN-K3 系列 PLC 的体系结构

1.1.1 综述

KDN-K3 是高品质的小型一体化可编程控制器，在 CPU 本体中集成了电源、通讯口和一定数量的 I/O，再通过扩展模块可以实现中小规模的控制系统。由于采用了软硬件优化设计，CPU 逻辑指令执行时间小于 0.5μs，为适应复杂的过程控制和机器控制需求，特别加入了软件 PID 算法、运动控制等多种高级控制指令。

KDN-K3 系列 PLC 的硬件包括 CPU 和扩展模块两大类，CPU 模块有 3 个型号 7 种规格，扩展模块有 20 多种规格。这两类模块可以灵活组合出适应绝大多数应用的完整自动化系统。在使用时 CPU 模块安装在最左端，扩展模块从右边的扩展接口连出。CPU 通过扩展总线控制这些模块的工作模式并进行数据交互，同时还向扩展模块提供工作电源。

KDN-K3 系列 PLC 组建的是开放性的系统，提供了各种常用的网络通讯接口以实现与其它系统、设备的互联。支持的通讯方式包括串行通讯、现场总线以及工业以太网等。

KDN-K3 系列 PLC 的编程软件是 EasyProg。EasyProg 符合 IEC61131-3 标准，目前支持梯形图和指令表两种编程语言。EasyProg 为用户提供了生成控制程序、配置系统硬件、读取诊断信息、在线监测变量、强制输出以及程序文档处理等各种功能。

1.1.2 所支持的系统规模

下表列出了各类 CPU 支持的最大点数和最大扩展模块数。另外，受总线供电的限制，各类 CPU 所带 DO 扩展模块中继电器型点数也有限制。**注意：表中所有的数据都指的是最大限制，并且各种限制条件必须同时满足！**

	DI	DO		AI	AO	扩展模块数
		全部	继电器			
CPU304	24	24	24	8	4	2
CPU306	64	64	32	16	16	4
CPU308	256	256	64	32	32	15

表 1-1　各类 CPU 支持的系统规模限制

1.2　CPU 提供的输出电源

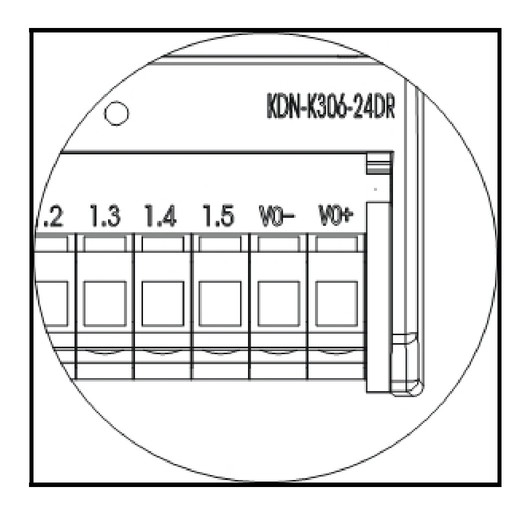
为提高系统集成度，方便用户的使用，KDN-K3 的 CPU 上设计了 DC24V 的电源输出，端子标号是 *VO+*、*VO-*。 CPU 以及所有扩展模块的输入通道都可以用此电源供电，其容量能够保证在 CPU 连接最大数量扩展模块的情况下也可以为所有输入通道供电。

各类 CPU 上输出电源的容量如下表：

CPU304	300 mA
CPU306	300 mA
CPU308	400 mA

表 1-2　CPU 集成输出电源的容量

下面图 1-1 是输出电源端子及其应用的示意图



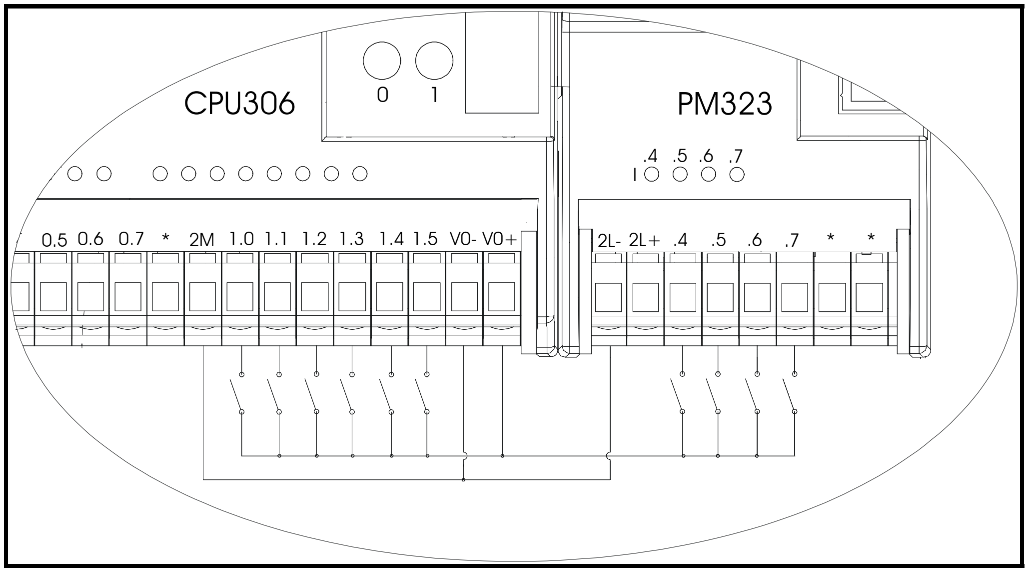


图 1-1 集成输出电源端子及应用示意图



VO+、VO-只建议为各模块的输入通道供电，不建议用做其他任何用途！

1.3 PLC 模块之间的连接

1.3.1 扩展总线概述

KDN-K3 的扩展总线是连接 CPU 模块和扩展模块的数据通道，在电气、机械上让各模块连接成一个整体的系统。CPU 对扩展模块的扫描是利用单独的通讯处理器进行的，扫描过程只占用很少的 CPU 资源。在连接最多 15 个扩展模块时，通讯处理器对全部模块的扫描时间不超过 1ms。

扩展总线的物理介质为 16 芯扁平电缆，在 16 芯中分别定义了两路高速数据通道、扩展模块地址译码通道、+5V 电源、电源地（GND）、+24V 电源等。扩展总线接口位于各模块的右端。

扩展总线提供的+5V 电源、+24V 电源均由 CPU 内部提供，其中+5V 电源是用于为各扩展模块内部电路提供工作电源，而+24V 电源是特别设计作为继电器型 DO 扩展模块中继电器线圈的驱动电源（DC24V）。CPU 对扩展总线中两种电源的供电电流如下表。

	+5V 电源	+24V 电源
CPU304	300mA	120mA
CPU306	720mA	165mA
CPU308	1200mA	240mA

表 1-3　扩展总线中+5V 和+24V 驱动电源的容量

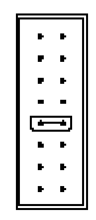
1.3.2　PLC 模块之间的连接

在实际的连接中，CPU 模块总是排列在最左端，扩展总线电缆从 CPU 的扩展接口引出，依次向右连接扩展模块，连接步骤为：将第一个扩展模块的 16 针扩展电缆插头接入 CPU 右端扩展接口的插座中，第二个扩展模块的 16 针扩展电缆插头接入第一个扩展模块右端的扩展接口插座，依此类推。扩展模块连接完成后将全部扩展模块推紧，扩展电缆自然滑入模块左侧的蔽线槽中，从正面看模块之间平滑过渡，没有缝隙。



图 1-2　PLC 模块连接后的实际图

另外，需要注意的是：CPU308 支持最多 15 个扩展模块。当扩展总线距离较长（超过 1 米）或者使用 CPU308 连接的扩展模块数量较多时，为增加扩展总线数据传输的稳定性，建议将最后一个模块扩展接口中的第 9 针与第 10 针使用短接跳线短接起来。如右图所示。



1.4　通讯网络接口

KDN-K3 系列 PLC 提供了多种网络通讯方式，能够方便地与第三方的设备、系统实现互联。使用 K3 系列 PLC 既可以满足单机设备的监控需要，更可以建立起从现场底层直至工厂管理层顺畅的网络通道，满足各类人员对监控信息的需要。

- 串行通讯

CPU 模块上集成的 RS232 和 RS485 接口支持标准 Modbus RTU 协议以及自由通讯方式。在默认设置下，使用的是 Modbus RTU 协议，K3 系列 CPU 作为 Modbus 从站。

KDN-K3 可以与任何支持标准 Modbus RTU 协议的人机界面互连。另外，对于诸如条码阅读器、智能仪表等使用自定义协议的串行设备，可以用自由通讯方式进行连接。

KDN-K3 也支持用 RS485 接口将最多 32 台 PLC 连成网络（使用 Modbus RTU 协议或者自定义协议）。



图 1-3　串行通讯示意图

▪ 现场总线

扩展功能模块 FM342 和 FM343 分别是 Profibus-DP 和 DeviceNet 的从站接口模块，能够将 K3 PLC 整体接入现场总线网络，网络速度达到标准支持的最高值。随着其他现场总线的普及，凯迪恩公司还将开发各种接口模块。

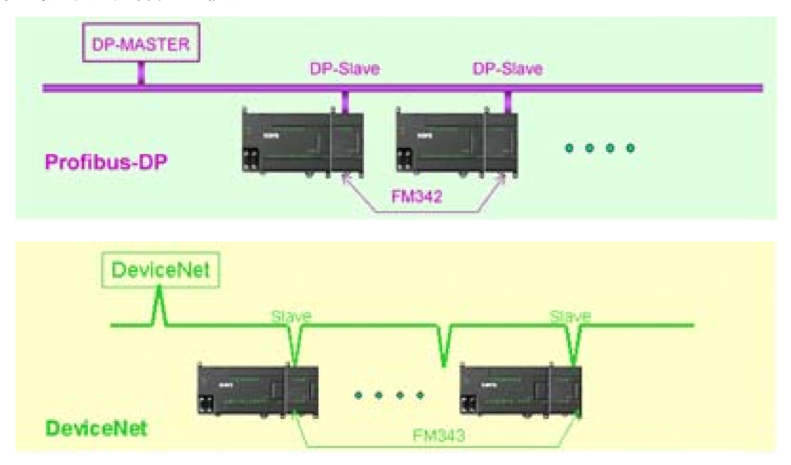


图 1-4　现场总线通讯示意图

▪ 以太网通信

KDN-K3 支持带宽达 100Mbps 的以太网，使用标准的 TCP/IP 协议，并提供 OPCServer 以接入 PC。凯迪恩公司可以向客户开放以太网协议，以利于客户二次开发。

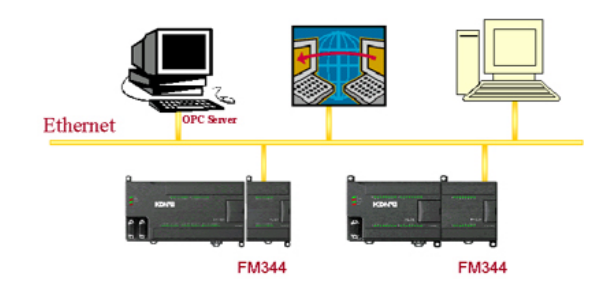


图 1-5　以太网通讯示意图

1.5　环境条件

以下环境参数适用于 KDN-K3 系列 PLC 中的所有产品。

工作温度	0 --- +55℃
允许相对湿度	95%，无冷凝
存储温度	-20 --- +85℃

第二章 CPU 原理及应用

本章详细描述了 KDN-K3 系列 PLC 中的 CPU 模块，介绍了硬件原理、原理框图、特殊功能、接线图、技术参数等信息。

2.1 概述

CPU 模块是 KDN-K3 PLC 的核心，通过扩展总线与扩展模块连接，构成一个完整的 PLC 硬件系统。CPU 模块负责执行“读入输入数据－>程序执行－>处理通讯请求－>自诊断－>写输出－>读入输入数据……”的工作循环处理，同时，控制扩展总线来完成对扩展模块的数据读取、数据输出。

KDN-K3 CPU 的系统程序（板级固件）存放在非易失闪烁存储器 FLASH 中，用于编译执行用户的应用程序。用户的应用程序经由上位编程软件（EasyProg）下载到 KDN-K3 CPU 后，存放在易失数据存储器 RAM 内，同时备份在铁电非易失存储器（FRAM）内（用于当 RAM 中的用户应用程序丢失时恢复程序）。CPU 掉电情况下 RAM 中的应用程序数据由超级电容来进行数据保持，可保证在 72 小时内数据不丢失。用户的应用程序由程序代码、配置数据两部分组成。

用户通过 EasyProg 进行硬件配置->程序编辑->程序编译->下载后，KDN-K3 PLC 即可按照程序要求完成用户的应用控制。

KDN-K3 PLC 具有结构紧凑、扩展性良好、价格低廉、指令强大的特点，丰富的 CPU 类型和多种供电电压等级使得在解决用户的工业自动化问题时，具有很强的适应性。

2.1.1 超级电容

CPU 本体提供一个超级电容，用于在 CPU 断电时保持 RAM 中的应用程序数据。用户可以有选择地设定需要保持的数据区类型及起止范围。通过 EasyProg “PLC 硬件配置－>保持区域”

选择需保持的数据区类型（如 V 区、M 区、C 区等）及起止范围，在 KDN-K3 CPU 重新上电后，被设定为保持的数据区将保持断电时的状态及数据值。

在 25℃环境温度下，超级电容保持用户程序数据时间为 72 小时；当断电时间超过 72 小时后，原设定的保持区中的数据将丢失。在使用超级电容保持程序数据前，需要连续给 CPU 上电超过 10 分钟以对超级电容充电。保持数据区类型及范围大小详细内容见本手册软件部分 4.7.3.1。

2.1.2 铁电非易失存储器（FRAM）

CPU本体提供一个铁电非易失存储器（FRAM）来永久保持用户的应用程序、配置数据。当RAM中用户应用程序、配置数据丢失时，CPU重新上电后系统会将FRAM中的应用程序、配置数据恢复到RAM中。铁电非易失存储器（FRAM）具有无限制的读写次数、10 年掉电数据保存期、无延时写操作等特点。

铁电非易失存储器（FRAM）的另一个功能是，在用户程序控制下，永久备份 V 区中特定范围（%VB3648~%VB3902）内 255 字节的数据。V 区需备份数据的内存地址存储在系统存储器 SMW32 内，此数值是距 V0 地址的偏移量；SM31.7 为使能写入控制标志位，CPU 系统软件实时读取 SM31.7 位状态，当为“TRUE”时， V 区相应内存地址内的数值被写入到铁电非易失存储器（FRAM）的相应位置；否则不执行。关于该功能的使用说明见附录 B 系统存储器 SMB31、SMW32。

2.1.3 实时时钟（RTC）

CPU本体内置一个实时时钟(RTC)，可提供实时时间/日历表示。实时时钟/日历的秒至年采用BCD格式编码，自动进行闰年调整，使用电容后备。电容后备的时间为72小时。可以通过EasyProg在线设置/读取实时时钟（RTC）操作。此外EasyProg还提供写/读取实时时钟（RTC）指令，实现与时钟有关的控制应用。

2.1.4 其它功能

CPU本体还提供如下的一些功能，关于这些功能详细的介绍请参见相关章节。

- 两个 10 位分辨率的模拟电位器（顶调电位器）；
- 128 个定时器；
- 128 个计数器；
- 一个 RS-232 串口，可用作编程口以及与第三方设备的通讯口；
- 6 个高速计数器，支持 12 种工作模式；
- 两路高速输出，支持 PTO/PWM 输出模式及对包络表编程；
- 中断检测；
- 支持 Modbus RTU 协议以及自由协议通讯。

2.2　CPU 型号及描述

CPU 模块共分三类：CPU304、CPU306、CPU308，编号越高则意味着此类 CPU 本体集成有更多 I/O 点数以及具有更高的性能。

CPU 本体集成有一定数量的 DI 点和 DO 点，其中 DI 是晶体管型输入，可以接源型/漏型输入信号；DO 是晶体管型或者继电器输出。

CPU 本体的 DI 输入既可用作普通的 DI 输入，也可用作高速脉冲输入、边沿捕捉。若 CPU 本体 DO 是晶体管类型的，则 Q0.0、Q0.1 两个通道即可用做普通的 DO 输出，也可以用做高速输出；若 DO 是继电器类型的，则 Q0.0、Q0.1 只能用做普通的 DO 输出。

CPU 的供电电源电压可以使用 DC 24V，也可以使用 AC 85V~265V。

2.3　CPU 结构

2.3.1 前面板

图 2-1 为 KDN-K306-24 DT CPU 移去端子盖板和扩展口盖板后的前面板正视图，包括：运行开关、顶调电位器、CPU 型号标签、信号线连接端子、I/O 状态指示灯、编程口、CPU 状态指示灯、扩展总线接口等几个部分。

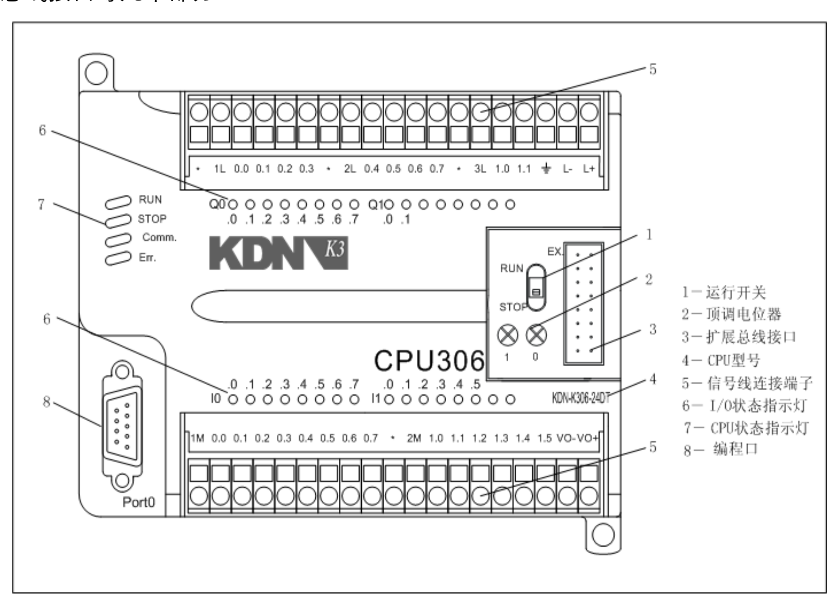
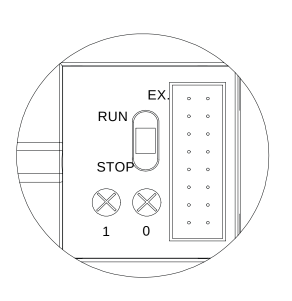


图 2-1　KDN-K306-24 DT CPU 前面板正视图

2.3.2 运行开关

运行开关是启动或停止 KDN-K3 CPU 的开关，包括运行（RUN）、停止（STOP）两个状态，如右图所示。



当运行开关处于运行（RUN）位置时，CPU 处于运行状态，同时通过扩展总线控制器来完成对扩展模块的数据读取、数据的输出控制。

当运行开关处于运行（RUN）位置时，可以通过 EasyProg 远程将 KDN-K3

PLC 由运行状态切换到停止 STOP）状态。

当运行开关处于停止（STOP）位置时， KDN-K3 PLC 处于停止状态， CPU 不执行工作循环，同时，扩展模块也处于停止状态。

在运行开关处于停止（STOP）位置时，CPU 将依据 EasyProg“PLC 硬件配置”－>“IO 配置－>停机保持”的配置要求，执行在停止状态下的保持输出。

无论 CPU 处于运行或停止状态下，EasyProg 均可与 CPU 本体进行通讯、程序下载、在线监视、调试程序等。

2.3.3　CPU 运行状态指示灯

CPU 运行状态指示灯包括运行（RUN）、停止(STOP)、通讯(Comm)、故障(Err)四个指示灯。如图所示。

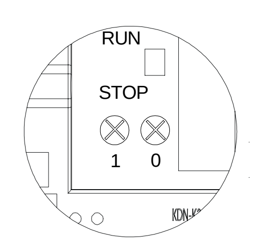
运行（RUN）、停止(STOP) 指示灯指示 CPU 的运行、停止状态，运行（RUN）指示灯为绿色，停止(STOP) 指示灯为红色。

通讯(Comm)指示灯用以指示 CPU 与 EasyProg 或第三方设备通讯时的状态。正常通讯时绿色通讯(Comm)指示灯会闪烁。

故障(Err)指示灯用以指示 CPU 在运行过程中发生通讯、程序运行错误、产生硬件故障等。故障(Err)指示灯为红色。当在运行中产生错误状态时，CPU 会依据错误状态的级别，来进行错误处理。如产生致命错误时，CPU 会立即停止执行正常的工作循环，同时将运行状态切换到停止状态，并点亮故障指示灯，提示用户排除故障。

2.3.4 顶调电位器

CPU 本体提供两个 10 位分辨率的顶调电位器，编号为 0、1。电位器的调节范围为 0~1023。用户可以使用螺丝刀来进行调节：顺时针旋转则增加顶调电位器数值，逆时针旋转则减少顶调电位器数值。



两个顶调电位器的数据值被送入系统存储器 SMW26、SMW28， SMW26 对应 0 号顶调电位器，SMW28 对应 1 号顶调电位器。

顶调电位器只能用做“只读”值使用，而不能改变它们的数值。顶调电位器的数值可以用做模拟量的输入值或输出值、定时器或计数器的当前值、预设值，或其它的中间值，从而方便用户的程序调试。

2.3.5　CPU 本体 I/O

CPU 本体的 I/O 地址是固定不变的，由 I/O 的类型及在 I/O 信号线连接端子中的位置决定的。

2.3.5.1　CPU 本体输入（DI）

输入（DI）部分在 CPU 本体的下侧。CPU306 提供 14 路 DI 通道，共分为两组：第一组包括 8 点，地址为 I0.0~I0.7；第二组包括 6 点，地址为 1.0~I1.5。各输入通道与内部 CPU 电路之间均有光电隔离，并对应相应的状态指示灯，指示输入通道的通断状态。

CPU 本体的 DI 输入端，既可以用于普通的数字量（开关量）输入，也可以用于高速脉冲量输入，各输入端的详细接入说明见本章 2.4.1 高速计数器功能部分。

DI 输入通道主要特点：

- 14 路晶体管输入通道，共分成 2 组，一组 8 个通道，另一组 6 个通道
- 固定输入地址：I0.0~I0.7，I1.0~I1.5
- 各组既可接源型输入（共阴极），也可以接漏型输入（共阳极）
- 额定输入电压为 DC24V，有效范围为 DC15～30V
- 现场信号与内部信号之间有光电隔离
- 各通道有独立的状态指示灯
- 既可以作普通数字量输入，也可作高速脉冲量输入

DI 端子接线图

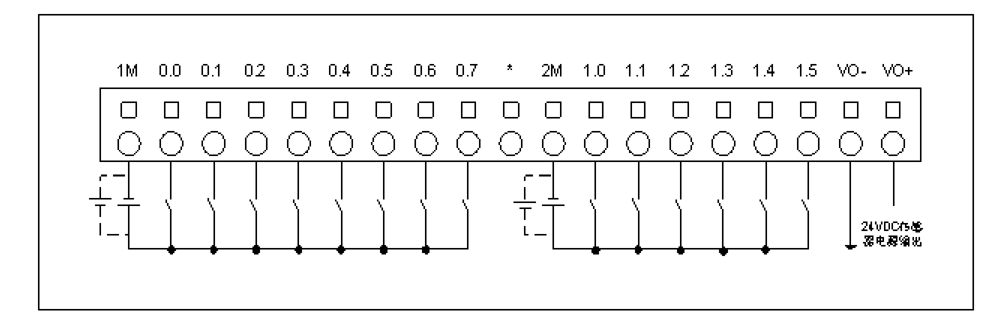


图 2-2　CPU306 的 DI 端子接线图

2.3.5.2　CPU 本体输出（DO）

输出（DO）部分在 CPU 本体的上侧。CPU306 提供 10 路 DO 输出，分成三组：第一组包括 4 路输出（DO），地址为 Q0.0~Q0.3；第二组包括 4 路输出（DO），地址为 Q0.4~Q0.7 地址；第三组包括 2 路输出（DO），地址为 Q1.0~Q1.1 地址。

各输出通道与内部 CPU 电路之间均经光电隔离，各 DO 输出端均有相应的状态指示灯，指示输出端子的通断状态。

晶体管型 DO 输出通道主要特点：

- 10 路晶体管输出通道，分成三组，第一组 4 路输出，第二组 4 路输出，第三组 2 路输出
- 额定供电电压为 DC 24V
- 额定输出电压为 DC 24V，每通道最大输出电流为 750mA，源型
- 感性负载输出保护
- 短路保护（每组输出电流大于 3A 时）
- 允许通道并联
- 输出与内部电路之间光电隔离
- 每通道独立发光二极管指示

继电器型 DO 输出通道主要特点

- 10 路继电器输出，分成三组，第一组 4 路输出，第二组 4 路输出，第三组 2 路输出
- 供电电压最高 DC30V/AC265V
- 每通道最大输出电流 3A（DC30V/AC265V）
- 每通道独立发光二极管指示

DO 端子接线图

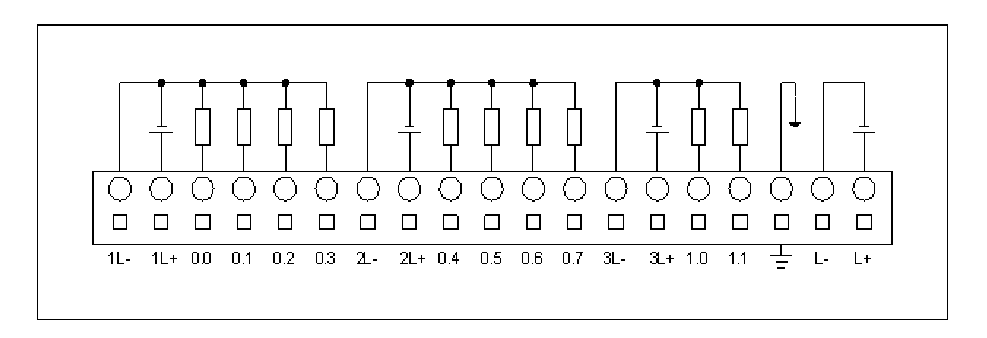


图 2-3　CPU306 的晶体管型 DO 端子接线图

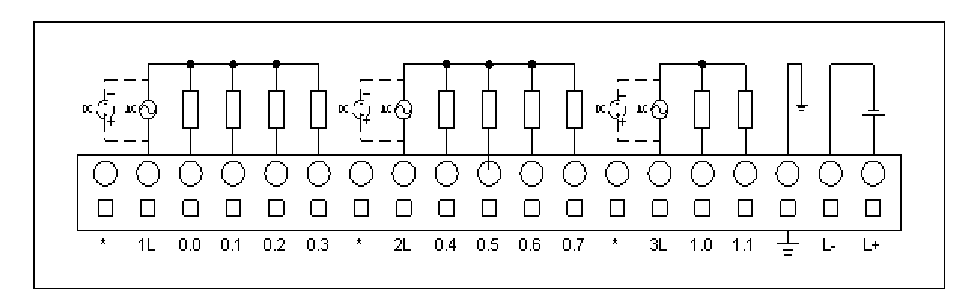
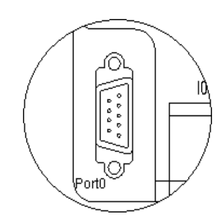


图 2-4　CPU306 的继电器型 DO 端子接线图

2.3.6 扩展总线接口

CPU 本体右部提供一个 16 针扩展总线接口用于连接扩展模块，详细描述请参见 1.3 PLC 模块之间的连接。

2.3.7 通讯口



CPU306 提供一个 RS-232 串行通讯口，采用 9 芯孔型 D 型口，如图所示。该口既可以用作编程口，也可以用作与第三方设备的通讯口。当采用屏蔽电缆时，串行通讯的最大距离为 15 米。

通讯口的信号定义如下表：

信号	符号	针号
信号地线	GND	5
发送数据	TXD	3
接收数据	RXD	2

表 2-1　通讯口信号定义

2.4　CPU 高级功能

CPU 本体可以完成一些高级功能，包括高速计数器功能、高速脉冲输出功能、边沿中断功能、串行口自由协议通讯功能。

高速计数器功能与高速输出功能结合起来，能构成完整的运动控制系统的闭环系统。 高速输出实现对运动控制的步进电机、伺服系统的控制，并通过高速计数器将当前的控制状态反馈回 CPU，来实现对运动控制系统的闭环控制。高速计数器、高速输出均支持多种中断处理功能，提供了丰富的运动控制指令，可以满足广泛的、不同行业的运动控制领域应用要求。

边沿中断功能可快速（纳秒级）捕捉到一个上升沿或下降沿的脉冲，并对于一些脉冲宽度小于 CPU 执行循环周期的输入信号可实现快速捕捉，以实现一些特殊应用。

CPU 本体的串行通信口可由用户通过自定义通讯协议与外部设备通讯，这种模式称为通讯口自由协议通讯模式，简称自由口通讯模式。自由口通讯模式为用户提供了一个自由选择通讯协议，与第三方设备进行通讯的能力，将大大提高用户的控制系统的开放性。第三方设备可以是 PC 机的通讯软件或变频器、PLC 等带 RS-232 串行通讯口的智能设备。

2.4.1 高速计数器功能

一般情况下，高速计数器输入端与一个轴式增量编码器的输出端连接，轴式增量编码器安装在一个固定轴上，该轴是一个步进电机或伺服系统的电机轴或有固定转速比关系的其它驱动轴。步进电机或伺服系统电机轴所转过的轴向角位移能通过轴式增量编码器产生相应数量的高速脉冲，这些高速脉冲被高速计数器的输入端捕捉到并进行计数，从而取得步进电机或伺服系统电机轴实际所转过的轴向角位移数据，为实现对步进电机或伺服系统输出控制提供数据。

KDN-K3 PLC 提供六个高速计数器，为 HSC0～HSC5,每个高速计数器允许的输入频率高达 30kHz。HSC3、HSC5 有一种工作模式，HSC0、HSC4 有七种工作模式，HSC1、HSC2 有 12 种工作模式。所有的高速计数器在相同的工作模式下有相同的功能。可以利用初次扫描系统存储器位 SM0.1 调用一个包含 HDEF 指令的子程序来定义高速计数器。

对高速计数器进行编程，需经过以下两步来实现：

1) 定义高速计数器（HDEF）：定义所采用高速计数器号（0～5）和模式（0～11），给出了高速计数器和计数模式之间的关系。
2) 执行高速计数器指令（HSC）:根据 HSC 指令的相应系统存储器（SM）位的状态，设置和控制高速计数器的工作模式。

高速计数器指令格式：

1) 定义高速计数器指令格式：HDEF HSC，MODE

   HSC：所指定使用的高速计数器号，为 0~5

   MODE：高速计数器所采用的工作模式，为 0~11

2) 高速计数器执行指令：HSC N

   N:指定了高速计数器号，为 0~5

   高速计数器指令格式的详细使用说明见本手册指令集部分。

2.4.1.1 高速计数器的控制位

HSC 0、HSC 1、 HSC 2、 HSC4 四个高速计数器有 3 个控制位，用来设置复位与启动输入

的有效电平，以及选择 1x 或 4x 计数方式（只能是正交计数器）。这些位在每个计数器的控制字节中，只有在执行 HSC 指令时才有用。这些位的定义见表 2-4。

在执行 HSC 指令前，必须把这些控制位设定到希望的状态，否则计数器对计数模式的选择取缺省设置。缺省的设置为：复位和启动输入高电平有效，正交计数速率是 4x（4 倍输入时钟频率）。一旦 HSC 指令被执行，就不能再更改计数器的设置。

HSC0	HSC1	HSC2	HSC4	描述
SM37.0	SM47.0	SM57.0	SM147.0	复位有效电平控制位：0=高电平有效；1=低电平有效
SM37.1	SM47.1	SM57.1	SM147.1	启动有效电平控制位：0=高电平有效；1=低电平有效
SM37.2	SM47.2	SM57.2	SM147.2	正交计数器计数速率选择：0=4x；计数率：1=1x 计数率

表 2-2 复位、启动和 1x/4x 控制位的有效电平

只有定义了高速计数器和所采用计数器模式，才能对高速计数器的动态参数进行编程。每个高速计数器都有一个控制字节。包括下列几项：3 个控制位、允许或禁止计数，计数方向控制（只能是模式 0，1，2）或对所有其它模式的初始化计数方向，及是否更新高速计数器当前值和预置值。执行 HSC 指令前，要确保控制字节和相关的当前值及预置值已被装载，表 2-3 对这些控制位逐一做了说明。

HSC0/4	HSC1/2	HSC3/5	描述
SM37.3/ SM147.3	SM47.3 /SM57.3	SM137.3/ SM157.3	计数方向控制位：0=减计数；1=增计数
SM37.4/ SM147.4	SM47.4/ SM57.4	SM137.4/ SM157.4	向 HSC 中写入计数方向：0=不更新；1=更新计数方向
SM37.5/ SM147.5	SM47.5/ SM57.5	SM137.5/ SM157.5	向 HSC 中写入预置值：0=不更新：1=更新预置值
SM37.6/SM147.6	SM47.6/ SM57.6	SM137.6 /SM157.6	向 HSC 中写入新的当前值：0=不更新；1=更新当前值

SM37.7/ SM147.7	SM47.7/ SM57.7	SM137.7 /SM157.7	HSC 允许：0=禁止 HSC：1=允许 HSC

表 2-3　HSC0～HSC5 的控制位

2.4.1.2 设定当前值和预置值

每个高速计数器都有一个 32 位的当前值和一个 32 位的预置值。当前值和预置值都是有符号整数。为了向高速计数器装入新的当前值和预置值，必须先设置控制字节，并把当前值和/或预置值存入特殊存储器字节中，然后必须执行 HSC 指令，从而将新的值送给高速计数器。表 2-4 为 HSC0、HSC1、HSC2、HSC3、HSC4 和 HSC5 的当前值和预置值。

要装入值	HSC0	HSC1	HSC2	HSC3	HSC4	HSC5
新当前值	SMD38	SMD48	SMD58	SMD138	SMD148	SMD158
新预置值	SMD42	SMD52	SMD62	SMD142	SMD152	SMD162

表 2-4　HSC0～HSC5 的当前值和预置值

2.4.1.3 高速计数器的状态字节

每个高速计数器都有一个状态字节，其中某些位指出了当前计数方向，当前值是否等于预置值，当前值是否大于预置值。表 2-5 对每个高速计数器的状态位作了定义。

HSC0/4	HSC1/2	HSC3/5	描 述
SM36.0/ M146.0	SM46.0/ SM56.0	SM136.0/ SM156.0	保留
SM36.1/ M146.1	SM46.1/ SM56.1	SM136.1/ SM156.1	保留
SM36.2/ M146.2	SM46.2/ SM56.2	SM136.2/ SM156.2	保留
SM36.3/ M146.3	SM46.3/ SM56.3	SM136.3/ SM156.3	保留
SM36.4/ M146.4	SM46.4/ SM56.4	SM136.4/ SM156.4	保留
SM36.5/ M146.5	SM46.5/ SM56.5	SM136.5/ SM156.5	当前计数方向状态位： 0=减计数；1=增计数
SM36.6/ M146.6	SM46.6/ SM56.6	SM136.6/ SM156.6	当前值等于预置值状态位： 0=不等；1=相等

SM36.7/ M146.7	SM46.7/ SM56.7	SM136.7/ SM156.7	当前值大于预置值状态位：0=小于等于；1=大于

表 2-5 HSC0～HSC5 的状态位

2.4.1.4 高速计数器输入线连接

表 2-6 给出了高速计数器时钟、方向控制、复位和启动所使用的输入。这些输入引脚的使用见表 2-6～表 2-13。

高速计数器	输入引脚
HSC0	I0.0,I0.1, I0.5
HSC1	I0.4,I0.7,I1.2, I1.3
HSC2	I0.6, I1.1, I1.4, I1.5
HSC3	I0.1
HSC4	I0.2, I1.0, I1.1
HSC5	I0.3

表 2-6 计数器的指定输入

各高速计数器的输入及与边沿捕捉中断功能的输入通道存在一些重叠，同一个输入不能同时用做两个高速计数器输入，也不能既用于高速计数器输入，又用于边沿捕捉中断功能。一个应用程序中，一个输入引脚只能用做一个输入功能，要么用于某一高速计数的输入，要么用于边沿捕捉中断。例如，如果 HSC0 采用模式 3，用到 I0.1 和 I0.5，I0.0 空闲，则 I0.0 可以用做边沿中断捕捉功能，也可以给 HSC3 使用。下表为高速计数器和边沿捕捉中断的输入分配表。

高速计数器	输入引脚													
	I0.0	I0.1	I0.2	I0.3	I0.4	I0.5	I0.6	I0.7	I1.0	I1.1	I1.2	I1.3	I1.4	I1.5
HSC0	√	√				√								
HSC1					√			√			√	√		
HSC2							√			√			√	√
HSC3	√													
HSC4			√						√	√				

HSC5				√										
边沿捕捉中断	√	√	√	√										

表 2-7 高速计数器和边沿捕捉中断的输入分配表

2.4.1.5 高速计数器工作模式

HSC 0、HSC 1、 HSC 2、 HSC 3、 HSC4、HSC5 高速计数器的工作模式描述、对应的输入管脚见下各表。

HSC　0				
模 式	描 述	I0.1	I0.0	I0.5
0	带内部方向控制的单相增/减计数器	时钟		
1			复位	
2			复位	启动
3	带外部方向控制的单相增/减计数器	时钟		方向
4			复位	方向
6	带增减计数时钟输入的双相计数器	时钟（减）	时钟（增）	
9	A/B 相正交计数器	时钟 B 相	时钟 A 相	

表 2-8　HSC 0 操作模式

HSC 1					
模 式	描 述	I0.4	I0.7	I1.2	I1.3
0	带内部方向控制的单相增/减计数器			时钟	
1		复位			
2		复位	启动		
3	带外部方向控制的单相增/减计数器			时钟	方向
4		复位			方向
5		复位	启动		方向

6	带增减计数时钟输入的双相计数器			时钟（减）	时钟（增）
7		复位			
8		复位	启动		
9	A/B 相正交计数器			时钟 B 相	时钟 A 相
10		复位			
11		复位	启动		

表 2-9　HSC 1 操作模式

HSC　2					
模 式	**描 述**	I0.6	I1.1	I1.4	I1.5
0	带内部方向控制的单相增/减计数器			时钟	
1		复位			
2		复位	启动		
3	带外部方向控制的单相增/减计数器			时钟	方向
4		复位			方向
5		复位	启动		方向
6	带增减计数时钟输入的双相计数器			时钟（减）	时钟（增）
7		复位			
8		复位	启动		
9	A/B 相正交计数器			时钟 B 相	时钟 A 相
10		复位			
11		复位	启动		

表 2-10　HSC 2 操作模式

HSC　3		
模式	**描述**	I0.1
0	带内部方向控制的单相增/减计数器	时钟

表 2-11　HSC 3 操作模式

<table>
<tr><th colspan="5">HSC　4</th></tr>
<tr><th>模 式</th><th>描 述</th><th>I0.2</th><th>I1.0</th><th>I1.1</th></tr>
<tr><td>0</td><td rowspan="3">带内部方向控制的单相增/减计数器</td><td rowspan="3">时钟</td><td></td><td></td></tr>
<tr><td>1</td><td>复位</td><td></td></tr>
<tr><td>2</td><td>复位</td><td>启动</td></tr>
<tr><td>3</td><td rowspan="2">带外部方向控制的单相增/减计数器</td><td rowspan="2">时钟</td><td></td><td>方向</td></tr>
<tr><td>4</td><td>复位</td><td>方向</td></tr>
<tr><td>6</td><td>带增减计数时钟输入的双相计数器</td><td>时钟（减）</td><td>时钟（增）</td><td></td></tr>
<tr><td>9</td><td>A/B 相正交计数器</td><td>时钟 B 相</td><td>时钟 A 相</td><td></td></tr>
</table>

表 2-12　HSC 4 操作模式

<table>
<tr><th colspan="3">HSC　5</th></tr>
<tr><th>模式</th><th>描述</th><th>I0.3</th></tr>
<tr><td>0</td><td>带内部方向控制的单相增/减计数器</td><td>时钟</td></tr>
</table>

表 2-13　HSC 5 操作模式

2.4.1.6 高速计数器的时序

为了更好地理解高速计数器的各工作模式，如下各图描述了高速计数器各工作模式时序。

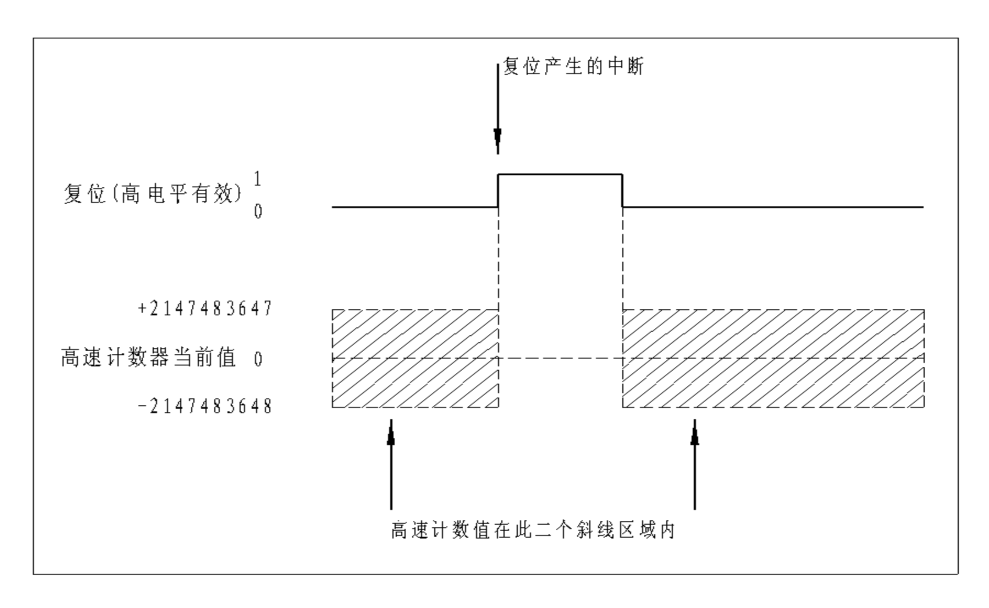


图 2-5　有复位无启动操作时序图

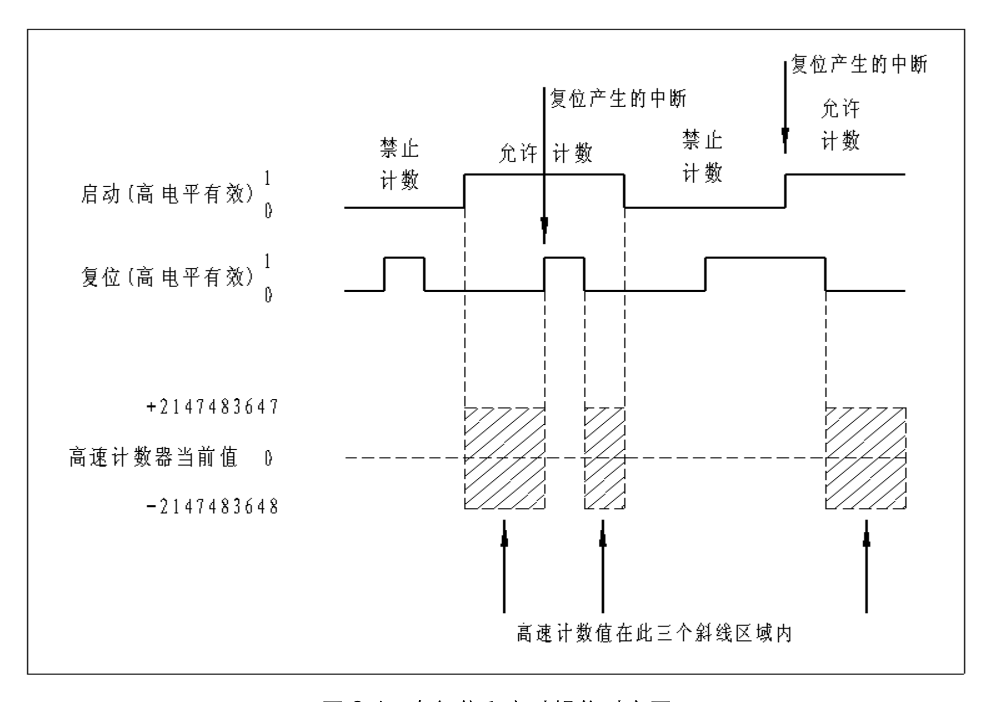


图 2-6　有复位和启动操作时序图

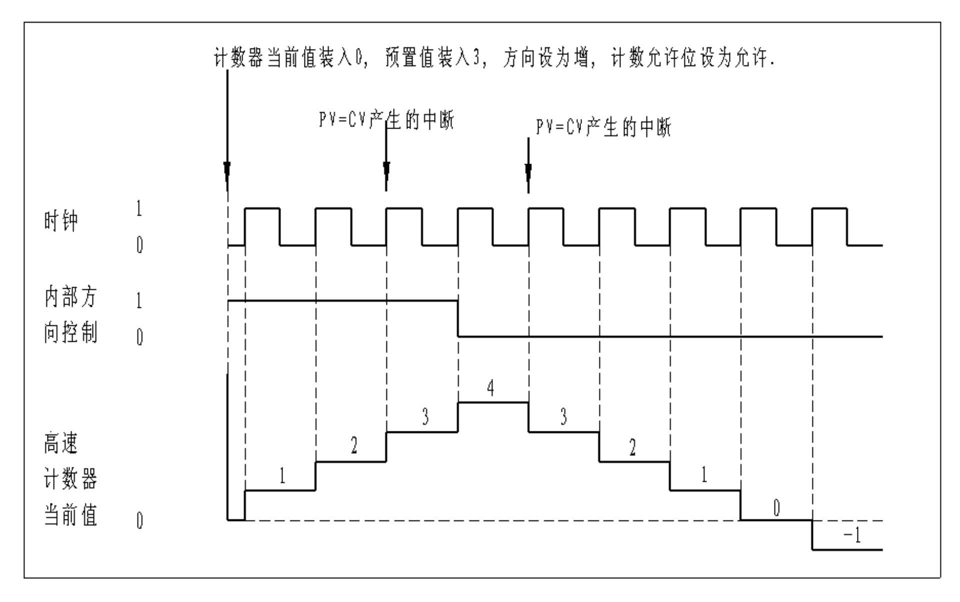


图 2-7　模式 0、1、2 操作时序图

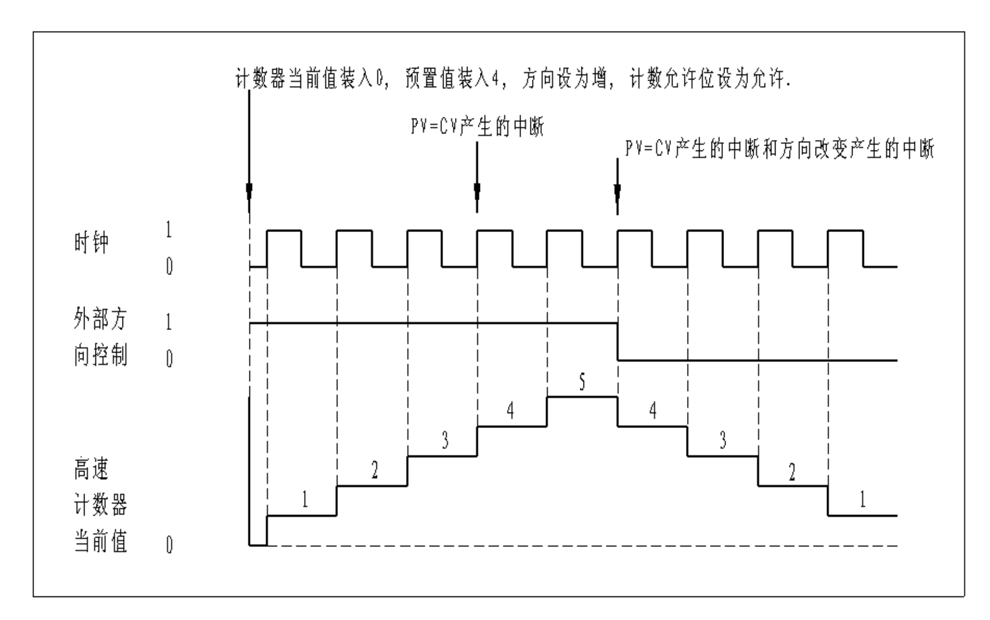


图 2-8　模式 3、4、5 操作时序图

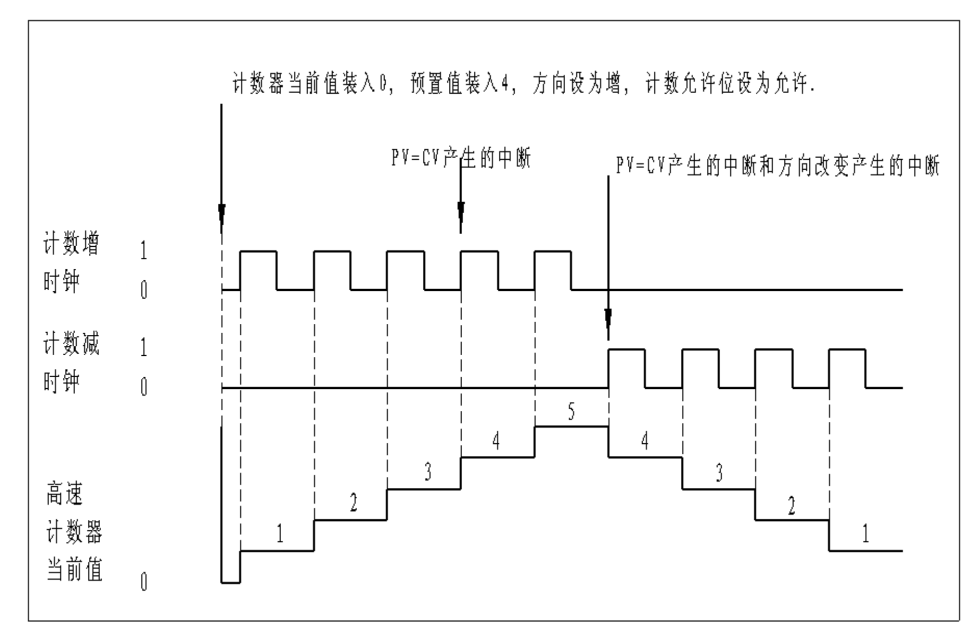


图 2-9 模式 7、8、9 操作时序图

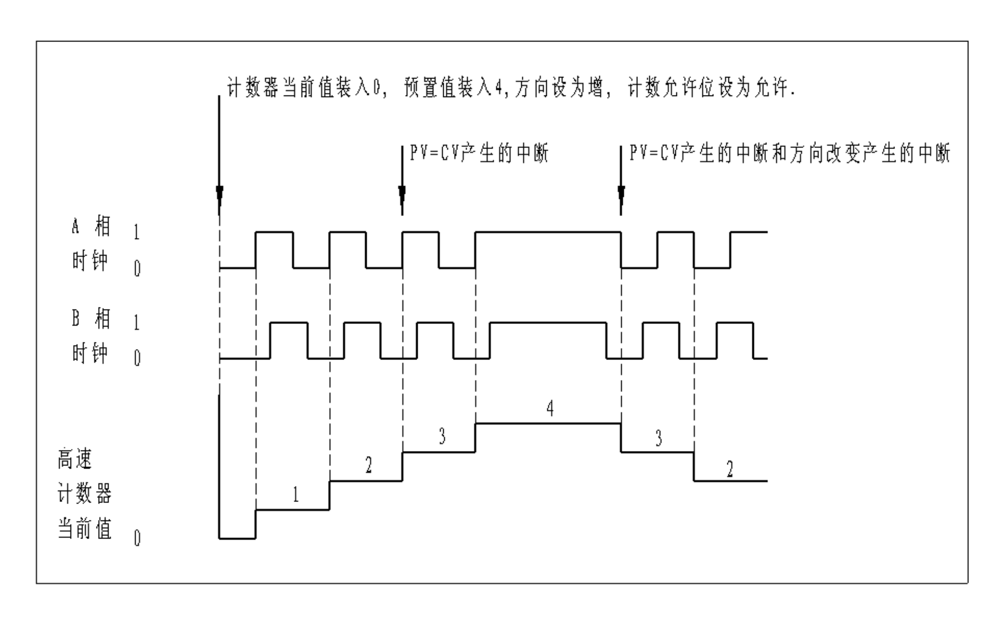


图 2-10 模式 9、10、11 正交 1×模式操作时序图

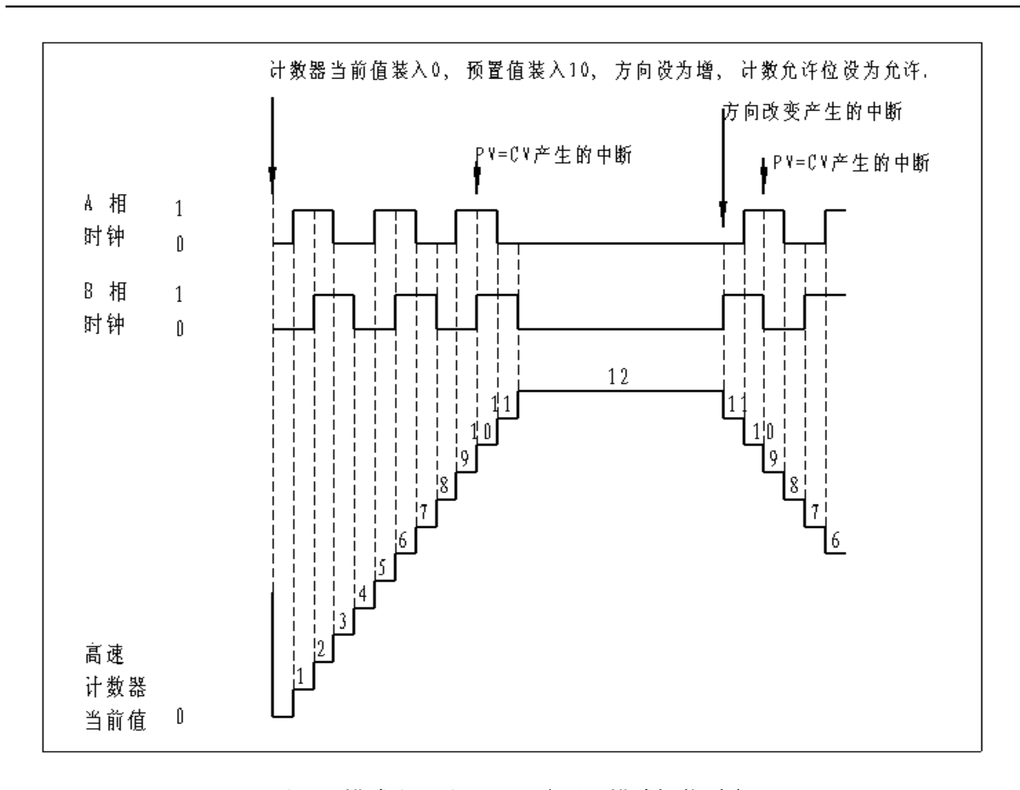


图 2-11 模式 9、10、11 正交 4×模式操作时序图

2.4.1.7 访问高速计数器

高速计数器的计数值当前值是只读的，用双字来（32 位）来寻址。高速计数器的当前值用 HC 加高速计数器号来区分，如 HC0 存取的是高速计数器 HSC0 的当前值。如图所示。

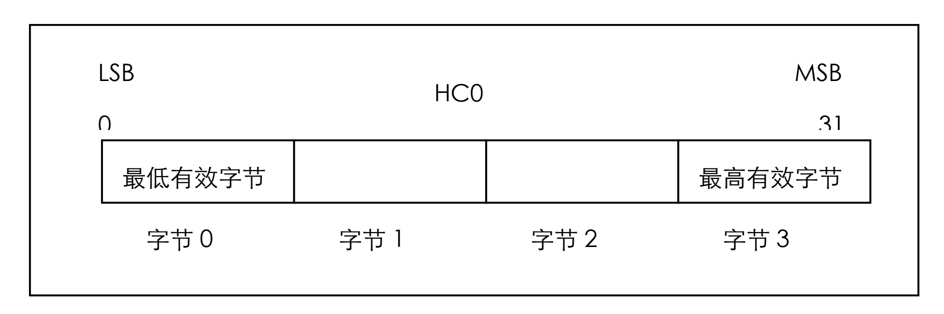


图 2-18 高速计数器的当前值格式

2.4.1.8 高速计数器编程

HSC0~HSC5 高速计数器均支持当前数值等于预设数值中断。使用外部复位输入计数器模式时，高速计数器支持外部复位有效时的中断。对每个高速计数器只能执行一次 HDEF 指令，如果对某高速计数器执行两次 HDEF，将生成运行时错误，而且并不改变第一次执行 HDEF 指令后该计数器的设定。

下面将以高速计数器 HSC2 为对模式 0～模式 11 的程序初始化作详细说明。

模式 0、1、2 的程序初始化：

1. 利用初次扫描存储器位 SM0.1 来直接初始化或调用一个包含初始化高速计数器子程序来初始化高速计数器。SM0.1 的使用说明详见附录 B 系统存储器 SMB0；

2. 初始化程序内，根据操作要求来设置 SMB57 的值。例如，SMB57 = b#16#F8 表示：
   - 高速计数器有效
   - 写入新当前数值
   - 写入新预设数值
   - 设定方向为增计数
   - 将启动及复位输入
   - 设定为高电平有效

3. 执行 HDEF 指令，HSC 取 2，MODE 取值：0（无外部复位或启动）、1（有外部复位、无启动）、2（有外部复位、启动）；

4. 将需设定的当前数值装载到 SMD58 (双字)，若装入 0，则清除 SMD58；

5. 将需设定的预设值装载到 SMD62 (双字)；

6. 捕捉当前数值等于预设值（CV=PV）中断，编写中断子程序，并将中断事件(事件 12)号与中断子程序绑定；

7. 为捕捉外部复位事件中断，编写中断子程序，并将中断事件(事件 10)号与中断子程序绑定；

8. 执行 HSC 指令，使 KDN-K3 PLC 对高速计数器进行程序处理。

模式 3、4、5 的程序初始化：

1. 利用初次扫描存储器位 SM0.1 来直接初始化或调用一个包含初始化高速计数器子程序来初始化高速计数器；

2. 在初始化程序内，根据操作要求为 SMB57 赋值。例如，SMB57 = b#16#F8 表示：

- 高速计数器有效
- 写入新当前数值
- 写入新预设数值
- 设定方向为增计数
- 将启动及复位输入
- 设定为高电平有效

3. 执行 HDEF 指令，HSC 值取 2，MODE 输入取 3（无外部复位或启动）、取 4（有外部复位、无启动）、取 5（有外部复位、启动）；

4. 将需设定的当前数值装载到 SMD58 (双字)，若装入 0，则清除 SMD58；

5. 将需设定的预设值装载到 SMD62 (双字)；

6. 为捕捉当前数值等于预设值（CV=PV）中断，编写中断子程序，并将中断事件(事件 12)号与中断子程序绑定；

7. 为捕捉方向改变中断，编写中断子程序，并将方向改变中断事件(事件 11)号与中断子程序绑定；

8. 为了捕捉外部复位事件中断，将外部复位中断事件(事件 10)号与中断子程序绑定，对中断处理进行编程；

9. 执行 HSC 指令，使 KDN-K3 PLC 对 HSC2 进行程序处理。

模式 6、7、8 初始化程序：

1. 利用初次扫描存储器位 SM0.1 来直接初始化或调用一个包含初始化高速计数器子程序来初始化高速计数器；

2. 初始化程序内，根据操作要求为 SMB57 赋值。例如，SMB57 = b#16#F8 表示：

- 高速计数器有效
- 写入新当前数值

- 写入新预设数值
- 设定方向为增计数
- 将启动及复位输入
- 设定为高电平有效

3. 执行 HDEF 指令，HSC 输入取 2， MODE 输入取 6（无外部复位或启动）、取 7（有外部复位、无启动）、取 8（有外部复位、启动）；

4. 将需设定的当前数值装载到 SMD58 (双字)，若装入 0，则清除 SMD58。

5. 将需设定的预设值装载到 SMD62 (双字)；

6. 为捕捉当前数值等于预设值（CV＝PV）中断，将中断事件(事件 12)号与中断子程序绑定，对中断处理程序进行编程；

7. 为捕捉方向改变中断，编写中断子程序，并将方向改变中断事件(事件 11)号与中断子程序绑定；

8. 为了捕捉外部复位事件中断，将外部复位中断事件(事件 10)与中断程序绑定，对中断处理进行编程；

9. 执行 HSC 指令，使 KDN-K3 PLC 对 HSC2 进行程序处理。

模式 9、10 或 11 初始化：

1. 利用初次扫描存储器位 SM0.1 来直接初始化或调用一个包含初始化高速计数器子程序来初始化高速计数器；

2. 初始化程序内，根据操作要求装载 SMB57。例如，

(1x 计数模式)

SMB57 = b#16#FC 表示：

- 高速计数器有效
- 写入新当前数值
- 写入新预设数值
- 设定方向为增计数
- 将启动及复位输入
- 设定为高电平有效

(4x 计数模式)

SMB47 = b#16#F8 表示：

- 高速计数器有效
- 写入新当前数值
- 写入新预设数值
- 设定方向为增计数
- 将启动及复位输入
- 设定为高电平有效

3. 执行 HDEF 指令，HSC 输入取 2， MODE 输入取 9（无外部复位或启动）、取 10（有外部复位、无启动）、取 11（有外部复位、启动）；

4. 将需设定的当前数值装载到 SMD58 (双字)，若装入 0，则清除 SMD58；

5. 将需设定的预设值装载到 SMD62 (双字)；

6. 为捕捉当前数值等于预设值（CV = PV）中断，将中断事件(事件 12)号与中断子程序绑定，对中断处理程序进行编程；

7. 为捕捉方向改变中断，编写中断子程序，并将方向改变中断事件(事件 11)号与中断子程序绑定；

8. 为捕捉外部复位事件中断，将外部复位中断事件(事件 10)与中断程序绑定，对中断处理进行编程；

9. 执行 HSC 指令，使 KDN-K3 PLC 对 HSC2 进行程序处理。

下面将以高速计数器 HSC2 为例，说明如何改变计数方向、装载新当前数值、装载新预设数值、关闭高速计数器。

改变模式 0、1、2 的计数方向：

下列操作说明如何设置 HSC2，使带内部方向(模式 0、1 或 2)的单相计数器改变方向。

1、 载 SMB57，写入所要方向：SMB57 = b#16#90 表示计数器有效，设定方向为减计数；

SMB57 = b#16#98 表示计数器有效，设定方向为增计数；

2. 执行 HSC 指令，KDN-K3 PLC 对 HSC2 进行程序处理。

装载新当前数值(在全部 12 种模式下)

以下操作说明如何改变 HSC2 的计数器当前数值：

1、装载 SMB47，写入所要当前数值：SMB47 = b#16#C0 计数器有效；

2. 用所要设定的当前数值装载 SMD58 (双字)，若装入 0，则清除 SMD58；

3. 将需设定的预设值装载到 SMD62 (双字)，若装入 0，则清除 SMD62；

4. 执行 HSC 指令，使 KDN-K3 PLC 对 HSC2 进行程序处理。

装载新预设数值(在全部 12 种模式下)

以下操作说明如何改变 HSC2 的计数器预设数值：

1. 装载 SMB57，写入所要预设数值：SMB57 = b#16#A0 计数器有效；

2. 用所要设定预设数值装载 SMD62；

3. 执行 HSC 指令，使 KDN-K3 PLC 对 HSC2 进行程序处理。

关闭高速计数器（在全部 12 种模式下)

以下操作说明如何关闭 HSC2 高速计数器:

1. 装载 SMB57，计数器无效：SMB57 = b#16#00 计数器无效；

2. 执行 HSC 指令，计数器无效。

上述操作说明如何逐一改变方向、当前数值以及预设数值，用户也可以按照相同操作，适当设定 SMB57 数值并执行 HSC 指令，改变全部数值或其中任何组合。

注意：关于高速计数器的编程实例见指令集计数器指令部分说明。

2.4.2 高速输出功能

KDN-K3 PLC 有两个 PTO/PWM 脉冲发生器产生高速脉冲串（PTO）或脉宽调制（PWM），输出脉冲频率可高达 20kHz。其中一个脉冲发生器分配给 Q0.0，称为 PWM0 或者 PTO0；另外一个分配给 Q0.1，称为 PWM1 或者 PTO1。

PTO/PWM 脉冲发生器和输出映像寄存器共同使用内存地址 Q0.0 和 Q0.1。当 Q0.0 或 Q0.1 设定为 PTO 或 PWM 功能时，PTO/PWM 发生器控制输出，并禁止通用功能的输出，包括强制输出功能、输出映像寄存器的状态。当不使用 PTO/PWM 发生器时，Q0.0、Q0.1 输出由输出映像

寄存器控制。

脉冲串（PTO）功能提供方波（50%占空比）输出，用户应用程序可控制输出脉冲周期和脉冲数。脉冲宽度调制（PWM）功能提供连续、变占空比脉冲输出，用户应用程序可控制输出脉冲周期和脉冲宽度。

每个 PTO/PWM 发生器有一个控制字节（8 位），各用一个 16 位无符号整型数来表示周期时间值和脉宽数值，用一个 32 位无符号双整型来表示脉冲计数值。这些值全部存储在指定的系统存储器（SM）中，一旦这些系统存储器按程序要求被设置完成，即可通过执行脉冲指令（PLS）来实现所期望的操作。

PLS 指令使 KDN-K3 PLC 读取系统存储器中的数据,并对相应的 PTO/PWM 发生器进行编程。修改系统寄存器（SM）区，然后执行 PLS 指令，可以改变 PTO 或 PWM 特性。把 PTO/PWM 控制字节（SM66.7 或 SM77.7）的允许位设置为 0，并执行 PLS 指令，可以在任何时候禁止 PTO 或 PWM 波形的产生。

所有控制字节、周期、脉冲数的缺省值都是 0。在 PTO/PWM 功能中，脉冲输出的 on 到 off 与 off 到 on 的切换时间不相同，PTO/PWM 的输出负载需要至少为 10%的额定负载，才能提供陡直的上升沿、下降沿。

2.4.2.1 PWM 操作

PWM 功能提供占空比可调的脉冲输出。周期和脉宽的增量单位为微秒(μs)或毫秒(ms)。周期变化范围分别为 50~65535 微秒或 2~65535 毫秒。脉宽变化范围分别为 0~65535 微秒或 0~65535 毫秒。当脉宽大于等于周期时，系统内部会自动将占空比设为 100%，输出连续接通。当脉宽为 0 时，占空比为 0%，即输出断开。

有两个方法改变 PWM 波形的特性：同步更新和异步更新。

・同步更新

在不改变时间基准前提下进行的更新。时间基准是指周期时间是选择微秒（μs ）还是毫秒（ms）作基准。利用同步更新，波形特性的变化发生在周期边沿，提供平滑转换。

·异步更新

PWM 的典型操作是当周期时间保持常数时变化脉冲宽度。如果需要改变 PTO/PWM 发生器的时间基准，就要使用异步更新。异步更新会造成 PTO/PWM 功能被瞬时禁止，会产生与 PWM 波形不同步，从而有可能会引起被控设备的振动。由于这个原因，建议采用 PWM 同步更新，即选择一个适合于所有周期时间的时间基准来进行同步更新。控制字节中的 PWM 更新 SM67.4 或 SM77.4 来指定更新类型，并通过执行 PLS 指令使这些有效。需注意的是，如果改变了时间基准，会产生一个异步更新，而和这些控制位无关。

2.4.2.2 PTO 操作

PTO 能产生指定脉冲个数的脉冲串（50%占空比）方波。周期以微秒（μs ）或毫秒（ms）为单位。周期的范围为 50 到 65535 微秒，或 2 到 65535 毫秒。如果设定的周期是奇数，会引起占空比的一些失真。脉冲数的范围是：1～4,294,967,295。如果指定脉冲数为 0，就把脉冲数缺省地设定为 1 个脉冲。

系统内存区 SM66.7 或 SM76.7 用来指示脉冲串输出是否已完成，完成时为“1”，否则为“0”。此外，脉冲串输出完成事件可以与一中断子程序绑定，当脉冲串输出一完成，即调用相应中断处理子程序。

PTO 功能支持单段、多段管线脉冲串输出。PTO 功能允许脉冲串排队，当已激活的脉冲串输出完成时，立即会开始新脉冲串输出。

单段管线

在单段管线中需要为下一个要输出的 PTO 脉冲串而更新特殊寄存器：一旦启动了起始 PTO 段，就必须立即按照第二个波形的要求改变特殊寄存器，并再次执行 PLS 指令，以此类推。第二个脉冲串的设定在管线中一直保持到第一个脉冲串发送完成。在管线中一次只能保存一个 PTO 段值，一旦第一个脉冲串发送完成，接着输出第二个脉冲串。重复该过程进行下一步脉站串的设定。

在下面的情况下，前后脉冲串之间的转换是不平滑的：

- 当发生时间基准的改变时；

- 当在利用 PLS 指令启动新脉冲串前，已启动的脉冲串已经完成。

除以上两种情况外，前后脉冲串之间的转换均是平滑的。当管线满时，如果试图装入新的管线，状态寄存器中的 PTO 溢出位（SM66.6 或 SM76.6）将置位。当 PTO 溢出位置位时，必须在程序中清除这个位。系统程序初始化完成后，PTO 溢出位（SM66.6 或 SM76.6）为 0。

多段管线

在多段管线中，CPU 自动从 V 储器区的包络表中读出多段管线的每个别脉串段的设定数值。在此模式下，只使用特殊寄存器区的控制字节和状态字节。

使用多段操作时，首先应在 SMB67（对应 PTO0）、SMB167（对应 PTO1）中设置时间基准，可以选择微秒或毫秒，然后在 SMW168（对应 PTO0）、SMW178（对应 PTO1）中装入包络表的起始位置。在包络表中的所有周期值必须使用同一个基准，并且在包络执行时不能改变。多段落操作使用 PLS 指令来启动。**需特别说明的是，包络表的起始位置必须为 V 区中的奇数地址，如 VB3001。**

多段管线的每段长度是 8 个字节，包括一个无符号整型（16 位）周期值、一个无符号整型（16 位）周期增量值和一个无符号双整型（32 位）脉冲计数值组成。包络表的格式如下表所示。

<table>
<tr><th>相对于包络表起始位置的字节偏移</th><th>长度</th><th>段 数</th><th>含 义</th></tr>
<tr><td>0</td><td>字</td><td></td><td>段数（1 到 64）</td></tr>
<tr><td>1</td><td>字</td><td rowspan="3">第 1 段</td><td>初始周期（2 到 65535 时基）</td></tr>
<tr><td>3</td><td>字</td><td>每个脉冲的周期增量
（–32767 到 32767 时基）</td></tr>
<tr><td>5</td><td>双字</td><td>脉冲数（1 到 4294967295）</td></tr>
<tr><td>9</td><td>字</td><td rowspan="3">第 2 段</td><td>初始周期（2 到 65535 时基）</td></tr>
<tr><td>11</td><td>字</td><td>每个脉冲的周期增量
（–32767 到 32767 时基）</td></tr>
<tr><td>13</td><td>双字</td><td>脉冲数（1 到 4294967295）</td></tr>
<tr><td>…</td><td></td><td>…</td><td>…</td></tr>
</table>

表 2-13 多段 PTO 操作的包络表格式

PTO 多段操作的一个特点是按照周期输入值自动增减周期：在周期增量区输入一个正值将增加周期；输入一个负值将减少周期；输入 0 值将不改变周期。当需要人为终止一个正进行中的 PTO 包络时，只需要把状态字节中的用户终止位（SM66.5 或 SM76.5）置为 1。

当 PTO 包络执行时，当前启动的段落数目保存在 SMB166（或 SMB176）中。表 2-13 为多段 PTO 操作的包络表格式。

包络表数值计算

PTO/PWM 脉冲发生器的多段包络表功能多应用在运动控制中，特别是对步进电机、伺服电机的控制。

下面将以一个对步进电机控制的具体应用示例来说明在实际应用中如何计算多段包络表。对步进电机的控制包括 3 段：第一段从起始（4k）输出 200 个脉冲后加速到一个固定电机转速（20k），演示如何加速步进电机；第二段在固定转速（20k）下输出 3000 个脉冲，演示如何以恒速控制步进电机；第三段从固定转速（20k）输出 200 个脉冲后减速到起始转速（4k），演示如何减速步进电机。见图 2-19。

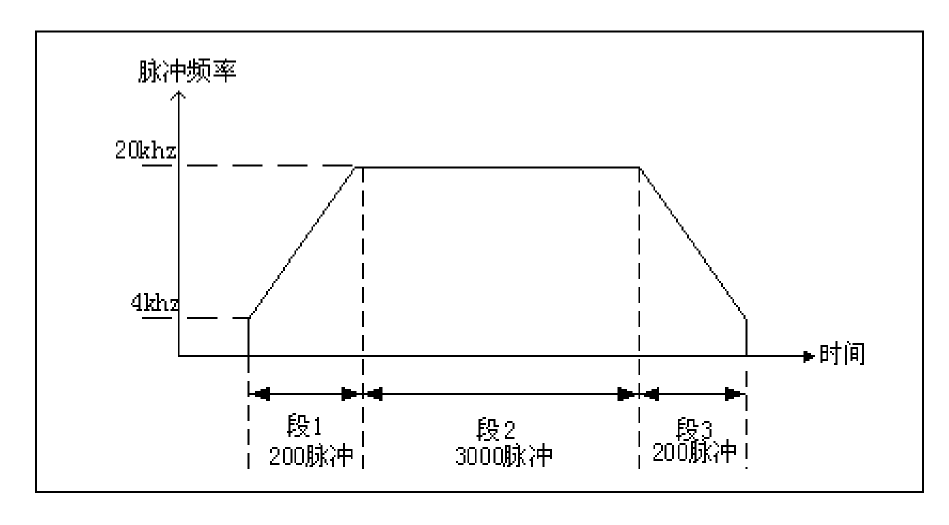


图 2-14 步进电机控制频率 – 时间图

在该示例中使用如下的计算公式来计算每列脉冲串周期的周期增量值：

（该段结束时的周期值 – 该段起始时的周期值）/该段脉冲数

在该示例中，第一段脉冲串的结束时的周期值：1/4*1000＝250 微秒；起始时的周期值：1/20*1000＝50 微秒；脉冲数：200；计算所得周期增量值：（250－50）/200=1 微秒。因第二段为恒速，周期增量值为 0；第三段的周期增量值为：-1 微秒。

假定包络表位于从 VB701 开始的 V 内存内(起始位置必须为 V 区开始的奇数地址,如 VB701、VB699 等)，表 2-15 是生成的包络表值：

V 区存储器地址	包络表值
VB701	3(总段数)
VW702	250 （起始周期，段 1）
VW704	-1 （周期增量，段 1）
VD706	200（脉冲数，段 1）
VW710	50 （起始周期，段 1）
VW712	0 （周期增量，段 1）
VD714	3000（脉冲数，段 1）
VW718	250 （起始周期，段 1）
VW720	1 （周期增量，段 1）
VD722	200（脉冲数，段 1）

表 2-15 多段包络表值

在多段包络表的各段之间实现平滑转换是非常重要的，平滑转换的条件必须是前一段的最后一个脉冲的周期值加上周期增量值等于下一段起始脉冲的周期值，为此，需要知道每段最后一个脉冲的周期值。每段最后一个脉冲的周期值采用如下公式计算：

段最后一个脉冲周期值＝该段起始周期值＋（该段周期增量值*（脉冲数-1））

如在本示例中计算第一段最后一个脉冲的周期值＝250＋（-1*(200-1)）=51 微秒

在确定包络表数值的过程中，知道一个包络段的持续时间是很用用的，计算公式入下：

包络段时间长度＝该段脉冲数*(该段起始周期值+(该段周期增量值/2)*(该段脉冲数-1))。如在本示例中计算第一包络段时间长度＝200*(250+ (1/2)*(200-1)) =19900 微秒。

2.4.2.3 PTO/PWM 寄存器

PTO0/PWM0 和 PTO1/PWM1 使用 SMB67/ SMB68 做控制字节，使用 SMW68/ SMW78、SMW70/ SMW80 做周期值、脉冲宽度值，使用 SMD72/ SMD82 做脉冲计数值，必须在执行 PLS 指令前装入以上这些值。使用 SMB166/ SMB176 存放当前执行的段数，使用 SMW168/SMW178 存包络表的起始位置。如果要使用多段脉冲串操作，在使用 PLS 指令前也需要装入包络表的起始偏移量（SMW168 或 SMW178）和包络表的值。下表是 PTO/PWM 脉冲发生器的控制寄存器表。

Q0.0	Q0.1	控制字节
SM67.0	SM77.0	PTO/PWM 更新周期值 0 – 不更新；1 – 更新周期值
SM67.1	SM77.1	PTO/PWM 更新脉冲宽度值 0 – 不更新；1 – 脉冲宽度值
SM67.2	SM77.2	PTO/PWM 更新脉冲数 0 – 不更新；1 – 更新脉冲数
SM67.3	SM77.3	PTO/PWM 时间基准选择 0 – 1 微秒/时基；1 – 1ms/时基
SM67.4	SM77.4	PTO 更新方法 0 – 异步更新；1 – 同步更新
SM67.5	SM77.5	PTO 操作： 0 – 单段操作；1 – 多段操作
SM67.6	SM77.6	PTO/PWM 模式选择 0 – 选择 PTO；1 – 选择 PWM
SM67.7	SM77.7	PTO/PWM 允许 0=禁止 PTO/PWM；1=允许 PTO/PWM
SMW68	SMW78	PTO/PWM 周期值（范围：2 到 65535）
SMW70	SMW80	PWM 脉冲宽度值（范围：0 到 65535）
SMD72	SMD82	PTO 脉冲计数值（范围：1 到 4294967295）
SMB166	SMB176	进行中的段数（仅用在多段 PTO 操作中）
SMW168	SMW178	包络表的起始位置，用从 V0 开始的字节偏移表示（仅用在多段 PTO 操作中）

表 2-16 PTO/PWM 控制寄存器表

下表是 PTO/PWM 脉冲发生器的状态寄存器，用以指示 PTO/PWM 脉冲发生器的状态。

Q0.0	Q0.1	状态字节
SM66.4	SM76.4	PTO 包络由于增量计算错误而终止 0=无错误；1=终止
SM66.5	SM76.5	PTO 包络由于用户命令而终止 0=无错误；1=终止

SM66.6	SM76.6	PTO 管线上溢/下溢 0=无上溢；1=上溢/下溢
SM66.7	SM76.7	PTO 空闲 0=执行中；1=PTO 空闲

表 2-17 PTO/PWM 状态寄存器表

2.4.2.4 PTO/PWM 程序编程

下面以 PTO0/PWM0 为例来介绍 PTO/PWM 脉冲发生器的程序初始化及编程。

PTO0 初始化 – 单段操作

按下列步骤实现对单段 PTO0 进行程序初始化：

1. 利用初次扫描存储器位 SM0.1 来直接初始化或调用一个包含初始化高速脉冲串（PTO0）输出子程序来初始化 PTO0。SM0.1 的使用说明详见附录 B 系统存储器 SMB0；

2. 在初始化程序或子程序内，以微秒为递增单位将数值 b#16#85 装载到 SMB67(以毫秒为单位装载值为 b#16#8D)中。产生下列结果：

- 启动 PTO/PWM 功能
- 选择 PTO 操作
- 选择以微秒或毫秒为递增单位
- 选择更新脉冲计数值及周期值。

3. 装载周期值到 SMW68 (字)；

4. 装载脉冲计数值到 SMD72 (双字)；

5. 如果需要将脉冲列完成事件(事件 28)与中断处理子程序绑定，可在此步利用 ATCH 指令执行绑定操作；

6. 执行 PLS 指令，使 KDN-K3 PLC 对 PTO/PWM 进行程序处理。

改变 PTO 周期时间 – 单段操作

按下列步骤可改变 PTO 周期时间值:

1. 为递增单位将数值 b#16#81 装载到 SMB67(以毫秒为单位装载值为 b#16#89)。产生如下结果：

- 启动 PTO/PWM 功能
- 选择 PTO 操作
- 选择以微秒或毫秒为递增单位
- 设定更周期值

2. 装载周期值到 SMW68 (字)；

3. 执行 PLS 指令，使 KDN-K3 PLC 对 PTO/PWM 进行程序处理。

改变脉冲数 – 单段操作

按下列步骤可改变 PTO 脉冲数值:

1. 以微秒为递增单位将数值 b#16#84 装载到 SMB67(以毫秒为单位装载值为 b#16#8C)。产生如下结果：

- 启动 PTO/PWM 功能
- 选择 PTO 操作
- 选择以微秒或毫秒为递增单位
- 设定更新脉冲计数

2. 装载周期值到 SMW68 (字)；

3. 执行 PLS 指令，使 KDN-K3 PLC 对 PTO/PWM 进行程序处理。

改变周期时间、脉冲数值 – 单段操作

按下列步骤可改变 PTO 脉冲数值周期时间、脉冲数值：

1. 以微秒为递增单位将数值 b#16#85 装载到 SMB67（以毫秒为单位装载值为 b#16#8d)。产生如下结果：

- 启动 PTO/PWM 功能
- 选择 PTO 操作

- 选择以微秒或毫秒为递增单位
- 设定新周期时间及脉冲数值。

2. 装载周期值到 SMW68 (字);

3. 装载脉冲计数值到 SMD72 (双字);

4. 执行 PLS 指令使 KDN-K3 PLC 对 PTO/PWM 进行程序处理。

PTO 初始化 – 多段操作

按下列步骤实现对多段 PTO0 进行程序初始化:

1. 利用初次扫描存储器位 SM0.1 来直接初始化或调用一个包含多段初始化高速脉冲串 0（PTO0）输出子程序来初始化 PTO0;

2. 在初始化程序或子程序内，以微秒为递增单位将值 b#16#A0 装载到 SMB67(以毫秒为单位值为 b#16#A8)。产生如下结果:

- 启动 PTO/PWM 功能
- 选择 PTO 及多段操作
- 选择微秒或毫秒为递增单位。

3. 将包络表的起始 V 内存偏移量地址（起始地址必须是奇数地址）装载到 SMW168 (字);

4. 设定包络表内的段数值（表内第一字节）;

5. 如果需要将脉冲列完成事件(中断时间号为 28)与中断处理子程序绑定，可在此步利用 ATCH 指令执行绑定操作;

6. 执行 PLS 指令，使 KDN-K3 PLC 对 PTO/PWM 进行程序处理。

PWM 初始化

按下列步骤实现对 PWM0 进行程序初始化:

1. 利用初次扫描存储器位 SM0.1 来直接初始化或调用一个包含初始化 PWM0 子程序来初始化 PWM0;

2. 在初始化程序或子程序内，以微秒为递增单位将值 b#16#D3 装载到 SMB67(以毫秒为递

增单位的值为 b#16#DB)。产生如下结果：

- 启动 PTO/PWM 功能
- 选择 PWM 操作
- 选择微秒或毫秒为递增单位
- 设定更新脉冲宽度及周期值。

3. 将周期值装载到 SMW68 中；
4. 将脉冲宽度值装载到 SMW70 中；
5. 执行 PLS 指令，使 KDN-K3 PLC 对 PTO/PWM 进行程序处理。

改变 PWM 输出脉冲宽度

按下列步骤实现改变 PWM 输出脉冲宽度(假定 SMB67 已被预先装载数值 b#16#D2 或 b#16#DA。)：

1. 将新的脉冲宽度值装载到 SMW70；
2. 执行 PLS 指令，使 KDN-K3 PLC 对 PTO/PWM 进行程序处理。

注意：关于高速输出的编程实例见指令集计数器指令部分说明。

2.4.3 边沿中断功能

边沿中断提供了能快速（纳秒级）捕捉到一个上升沿或下降沿的脉冲的能力。在 KDN-K3 PLC 中，CPU 本体的 I0.0~I0.3 输入可用做边沿中断功能。I0.0~I0.3 中的每一输入既可用做上升沿边沿捕捉中断，也可用做下降沿边沿捕捉中断，但在一个应用程序中，一个输入只能用做上升沿或下降沿边沿捕捉中断。每一输入的上升沿、下降沿边沿捕捉中断都有各自的中断号，将该中断号与中断子程序绑定，即可完成相应的边沿捕捉中断处理。当 I0.0~I0.3 做边沿捕捉中断功能时，其普通输入功能将不起作用。中断事件列表请参见本手册第三部分中 5.1.3.2 中断事件列表。

对于一些脉冲宽度小于 CPU 执行循环周期的输入信号可很快捕捉到，从而扩展了用户的控制系统的输入应用范围和特殊应用，如当某个时间发生时必须引起注意的异常输入等。

2.4.4 自由通讯功能

自由通讯为用户提供了与第三方带 RS 232 串行口的智能设备进行自由协议通讯的功能。通讯协议可以自行定义，从而大大提高用户的控制系统的开放性。第三方设备可以是 PC 机上的通讯软件或变频器、PLC 等带 RS-232 串行口的智能设备。

在自由通讯模式下，KDN-K3 PLC 可以实现数据接收、发送功能，以及与数据接收、发送完成绑定的中断，使得 KDN K3 PLC 有很强自由口编程能力、灵活放入处理功能，可以使用户更大范围地满足工业现场对自由口通讯的要求。选择自由口通讯模式后，用户就可以完全控制通讯端口的操作，通讯协议也完全受用户的程序控制。

2.4.4.1 自由通讯实现

通讯参数（比如波特率、奇偶校验等）在 EasyProg 的【PLC 硬件配置】中的 CPU 模块的〔通讯设置〕中来设置。将应用程序下载到 CPU 中，即可让设置的通讯参数生效。

若要使用自由通讯，则需要用到 SM 区中一些由系统预定义好的控制位、字节、字。另外，自由通讯的指令是 XMT（发送）和 RCV（接收）指令，请参见第三部分 6.11 通讯指令 中的详细说明及例子。

在自由通讯中提供“接收完成中断”、“发送完成中断”两种中断事件，可以使用中断来进行接收、发送完成后的后续处理。

2.5 硬件原理

KDN-K3 系列 PLC 的 CPU 采用了一款功能强大的 16 位 CPU 处理器，外接系统程序存储器（FLASH)、用户程序存储器(FRAM)、数据存储器（RAM)、实时时钟、电源电路、看门狗、输入/输出处理模块等构成基本的微计算机系统，执行 CPU 本体基本的输入、输出、定时器、计数器等程序处理及输入输出控制。数据存储器（RAM）外接超级电容器，在 25℃环境条件下可保持数据存储器（RAM）中的数据内容达 72 小时，以最大限度满足用户对数据存储器（RAM）中数据的保持要求。下图为 CPU 硬件原理框图。

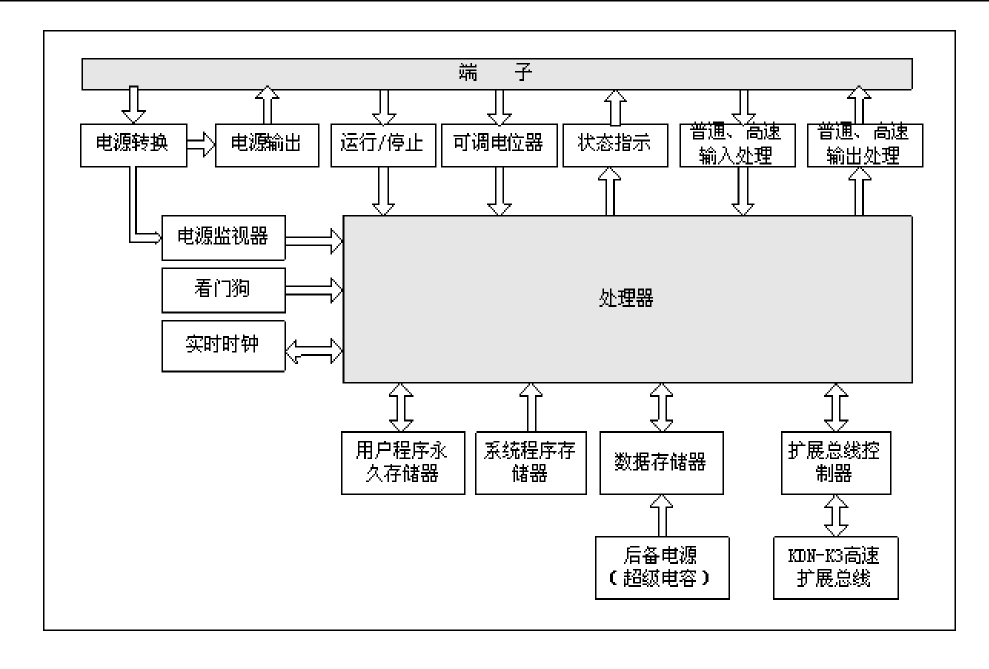


图 2-14　CPU 硬件原理框图

2.6 技术参数

2.6.1 CPU306 技术参数

性能指标	KDN-K306-24DT*	KDN-306-24DR	KDN-306-24AR
输入电源	DC24V±20%		AC85V-265V
本体输入点数	14×DC24V		
本体输出点数	10×DC24V 晶体管	10×继电器	
程序执行时间	布尔指令执行时间： 0.48μS 字操作时间：<48μS 整型数算术运算时间：65<μS 浮点数算术运算时间：<150μS		
应用程序存储器	4KB/约 1000 步		
程序备份	永久存贮，无需电池		

内存区域	数据存储区（V 区）：4KB 系统内存区（SM 区）：300 字节 中间变量区(M 区)：32 字节
数据保持	超级电容
数据保持时间	25℃时保持时间不低于 72 小时
支持的编程语言	梯形图（LD）、指令表（IL）
指令集	基本指令：50 条 扩展指令：252 条
口令保护	三级口令保护
可连接扩展模块数	4 个，不分种类
最大 I/O 数量	开关量：总计 88（DI、DO 各 64）； 模拟量：总计 20（AI、AO 各 16）
程序数据保持	通过配置软件可选
计数器	128
计数器工作模式	加、减、双向
计数范围	-32768~32767
定时器	128 1ms 时基：4 个 10ms 时基：16 个 100ms 时基：108 个
高速计数器	6 个高速计数器 其中：单相 6 个，最大 30KHz； 双相 2 个，最大 20KHz。
脉冲输出	2 个，最大 20KHz； 脉冲串输出（PTO），频率可变； 脉宽调制输出（PWM）；
顶调电位器	2 个，10 位分辨率，数值与内部寄存器对应
程序结构	一个主程序块（可以包括子程序）
程序执行	主循环，中断控制，定时控制
子程序数量	最大 32 个
中断功能	定时中断：2 个，1ms 分辨率； 事件中断：4 个，上升沿或下降沿可选； 通讯中断：2 个。

实时时钟	有，误差不大于 2 分钟/月@25℃ 可通过软件设置/读取：年、月、日、时、分、秒、星期
通讯口与通讯协议	1 个，RS232；Modbus RTU Slave、自由通讯。
输出电源	24V，容量 300mA

2.6.2　CPU 模块 DI 通道性能指标

CPU 模块输入特性	
输入类型	源型/漏型可选
额定输入电压	24VDC
额定输入电流	4.1mA@24VDC
最大输入电压	30VDC
逻辑 1 最小输入电压	15V，2.1mA
逻辑 0 最大输入电压	5V，0.7mA
输入滤波时间（非高速输入）	0.2~3 毫秒可选
输入延迟时间	输入接通延迟时间：5μs 输入断开延迟时间：8μS
输入与内部逻辑电路的隔离 • 隔离方式 • 隔离电压	 隔离方式：光电耦合器 隔离电压：1500VAC/1 分钟

2.6.3 CPU 模块晶体管型 DO 通道性能指标

输出类型	源型
额定输出电压	24VDC
额定输出电流	0.75A，每组最大输出电流：3A
抗浪涌电流	10A，0.1 秒
输出漏电流	25μA（最大）
导通阻抗	0.2Ω（最大）
输出接通延迟时间 高速输出（Q0.0、Q0.1） 标准输出	 高速输出（Q0.0、Q0.1）：7μS 标准输出：35μS
输出断开延迟时间 高速输出（Q0.0、Q0.1） 标准输出	 高速输出（Q0.0、Q0.1）：22μS 标准输出：170μS
输出与内部逻辑电路的隔离 • 隔离方式 • 隔离电压	 光电耦合器 1500VAC/1 分钟
感性负载输出保护功能	有
通道并联功能	有
短路保护功能	有

2.6.4 CPU 模块继电器型 DO 通道性能指标

输出类型	继电器
供电电源电压	最高 DC 30V/AC270V
额定输出电流	3A(DC 30V/AC270V)
输出电压范围	5~30VDC/5~270VAC
每组最大输出电流	10A
最大导通阻抗	0.1Ω（初始值）
输出接通延迟时间	5mS（典型值）
输出断开延迟时间	2mS（典型值）
最大开关频率 • 空载 • 额定负载	 12,000 次/分钟 100 次/分钟
触点预期寿命 • 机械寿命（空载） • 电气寿命（额定负载）	 20,000,000 次 100,000 次
隔离特性 • 隔离方式 • 线圈与接点的隔离电压 • 接点与接点的隔离电压	 继电器 线圈与接点的隔离电压：2000Vrms 接点与接点的隔离电压：750Vrms

2.7 安装尺寸

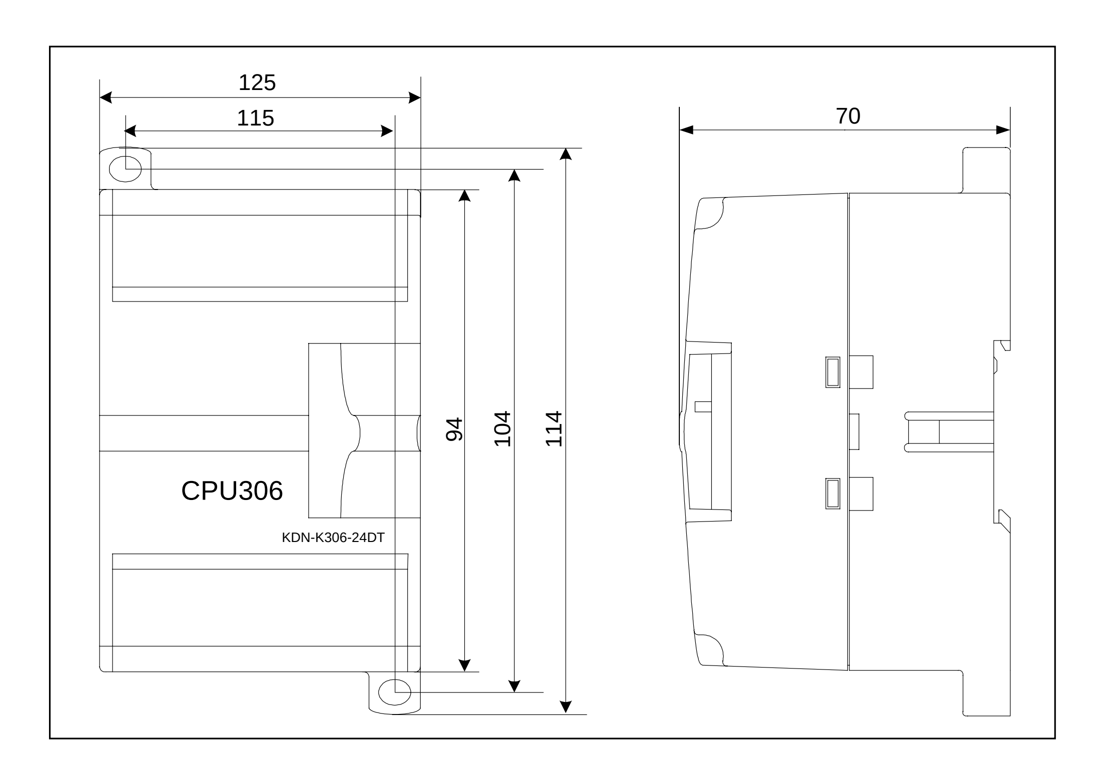


图 2-15　CPU 模块安装尺寸图

第三章 DI 扩展模块

本章详细描述了 KDN-K3 系列 PLC 中的 DI 扩展模块。每种模块均用独立一节详细介绍硬件原理、接线图、技术参数等信息。

DI 扩展模块类统一称为 PM321。

3.1 DI 8×DC24V

该模块的订货号是：KDN-K321-08DX。

这是具有 8 个通道的晶体管型开关量输入模块，基本功能是接收开关量信号输入并通过扩展总线将信号状态传送至 CPU 的 DI 区中。它在输入信号和内部电路之间提供了光电隔离。

该模块的各个通道均有指示其输入状态的指示灯。

3.1.1 主要特点

- 8 个输入通道，共分成 2 组，每组 4 通道；
- 各组既可以接源型输入（共阴极），也可以接漏型输入（共阳极）；
- 额定输入电压 DC24V，有效电压范围为 15～30V；
- 现场信号与内部电路之间光电隔离；
- 每通道独立发光二极管指示；
- 模块宽度 50mm。

3.1.2 前面板示意图

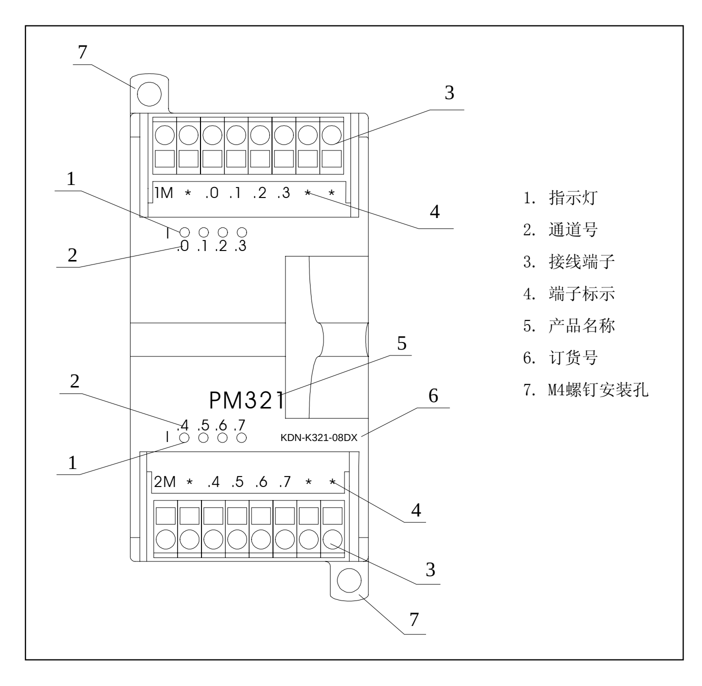


图 3-1　KDN-K321-08DX 前面板图

3.1.3　接线图和电气原理图

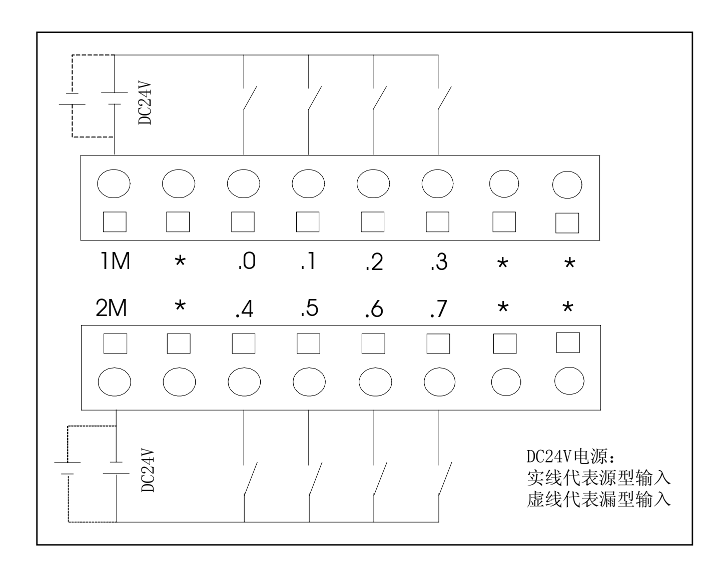


图 3-2 KDN-K321-08DX 接线图

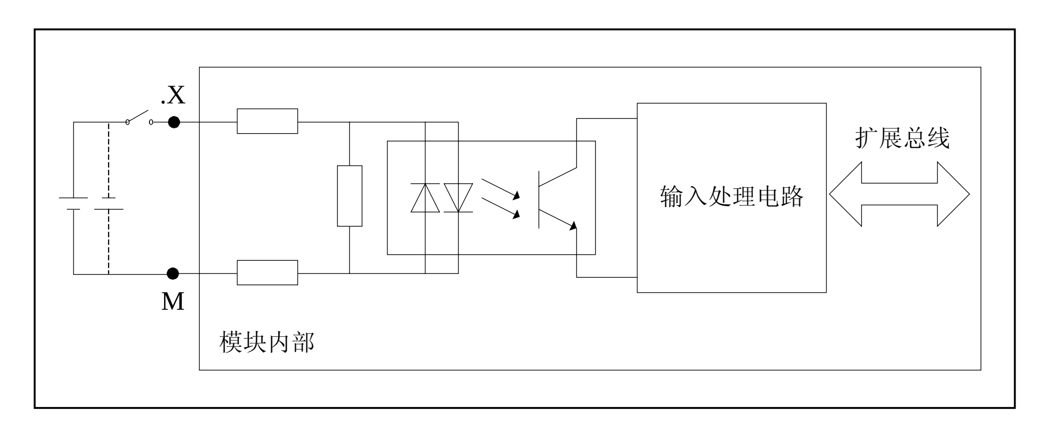


图 3-3 KDN-K321-08DX 电气原理图

3.1.4 安装尺寸图

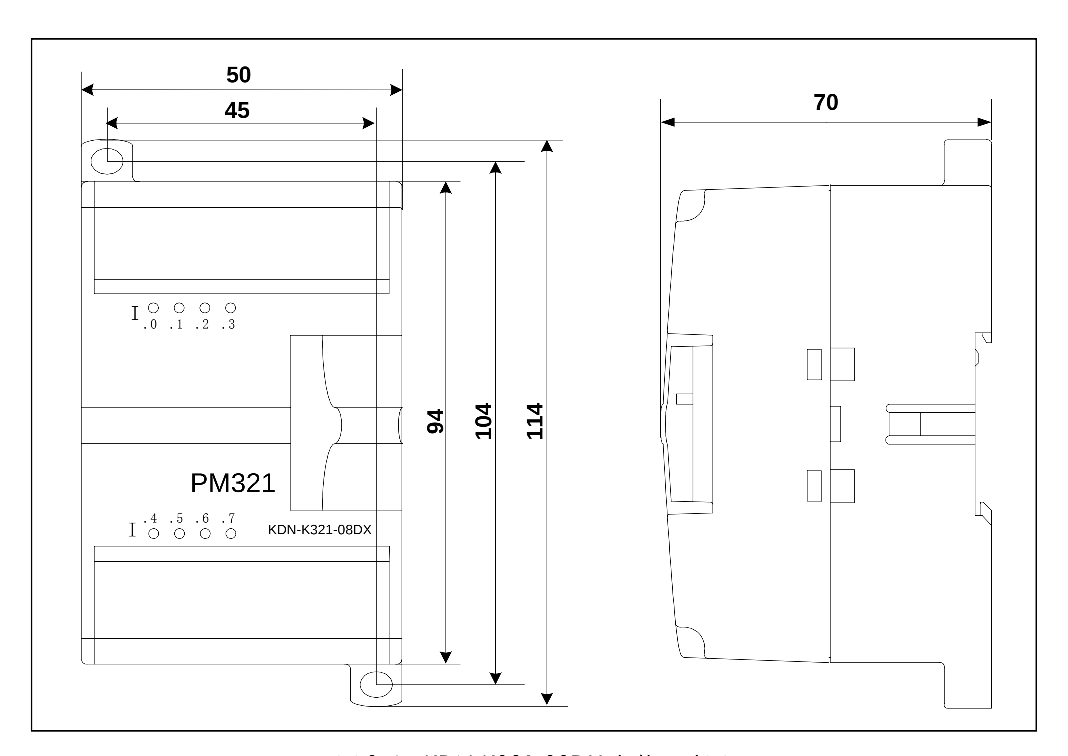


图 3-4 KDN-K321-08DX 安装尺寸图

3.1.5　技术数据

<table>
<tr><th colspan="3">电气参数</th></tr>
<tr><td>输入通道数</td><td colspan="2">8（4 通道/组）</td></tr>
<tr><td>输入类型</td><td colspan="2">源型/漏型</td></tr>
<tr><td>额定输入电压</td><td colspan="2">DC 24V（DC15～30V 时为“1”）</td></tr>
<tr><td>额定输入电流</td><td colspan="2">4.1mA@24VDC</td></tr>
<tr><td>逻辑“0”最大输入电压</td><td colspan="2">5V@0.7mA</td></tr>
<tr><td>逻辑“1”最小输入电压</td><td colspan="2">15V@2.5mA</td></tr>
<tr><td>输入滤波延迟</td><td colspan="2">5ms</td></tr>
<tr><td rowspan="2">扩展总线电流损耗</td><td>5V</td><td>72mA</td></tr>
<tr><td>24V</td><td>-</td></tr>
<tr><td>输入与内部逻辑电路的隔离
· 隔离方式
· 隔离电压</td><td colspan="2">
光电耦合器
1500VAC/1 分钟</td></tr>
<tr><td>状态指示</td><td colspan="2">绿色 LED 指示每个通道的信号“1”</td></tr>
<tr><th colspan="3">占用地址空间</th></tr>
<tr><td>DI 映像区</td><td colspan="2">1 字节</td></tr>
<tr><td>DO 映像区</td><td colspan="2">-</td></tr>
<tr><th colspan="3">尺寸和重量</th></tr>
<tr><td>尺寸(长×宽×高)</td><td colspan="2">114×50×70mm</td></tr>
<tr><td>净重</td><td colspan="2">125g</td></tr>
</table>

3.2 DI 16×DC24V

该模块的订货号是：KDN-K321-16DX。

这是具有 16 个通道的晶体管型开关量输入模块，基本功能是接收开关量信号输入并通过扩展总线将信号状态传送至 CPU 的 DI 区中。它在输入信号和内部电路之间提供了光电隔离。

该模块的各个通道均有指示其输入状态的指示灯。

3.2.1 主要特点

- 16 个输入通道，共分成 2 组，每组 8 通道；
- 各组既可以接源型输入（共阴极），也可以接漏型输入（共阳极）；
- 额定输入电压 DC24V，有效电压范围为 15～30V；
- 现场信号与内部电路之间光电隔离；
- 每通道独立发光二极管指示；
- 模块宽度 75mm。

3.2.2 前面板示意图

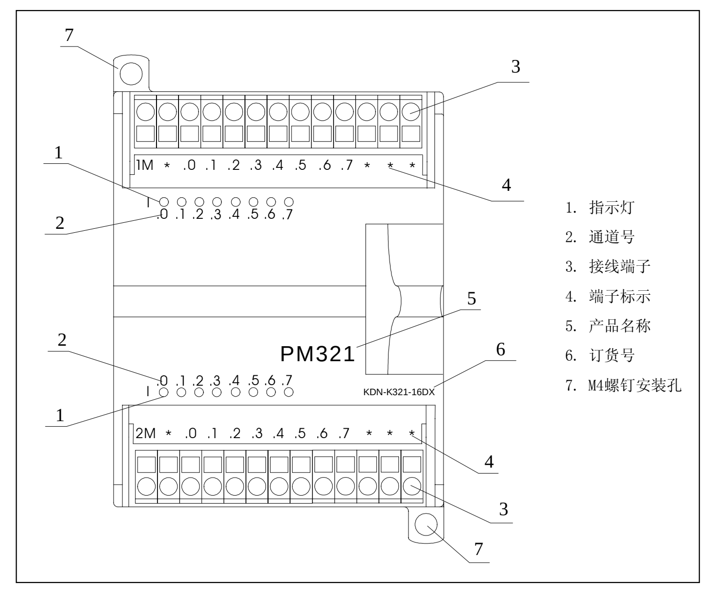


图 3-5　KDN-K321-16DX 前面板图

3.2.3　接线图和电气原理图

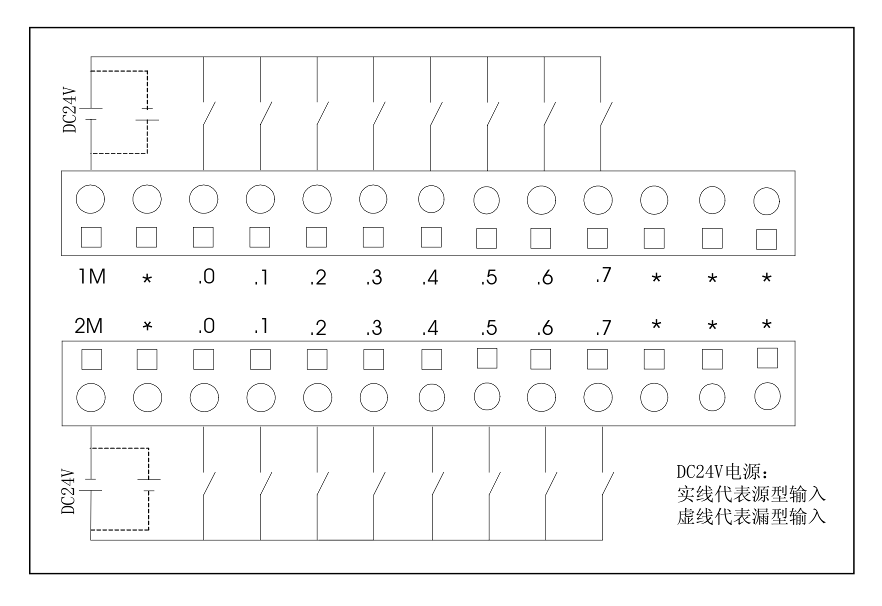


图 3-6　KDN-K321-16DX 接线图

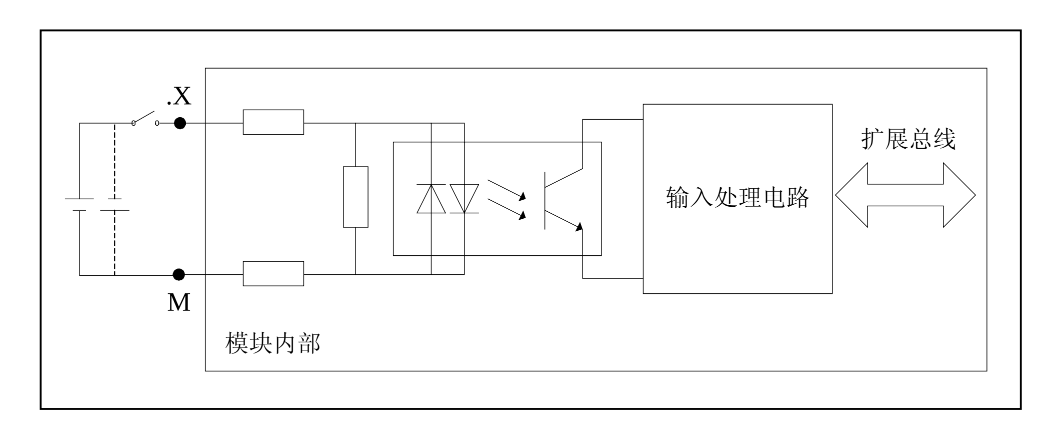


图 3-7 KDN-K321-16DX 电气原理图

3.2.4 安装尺寸图

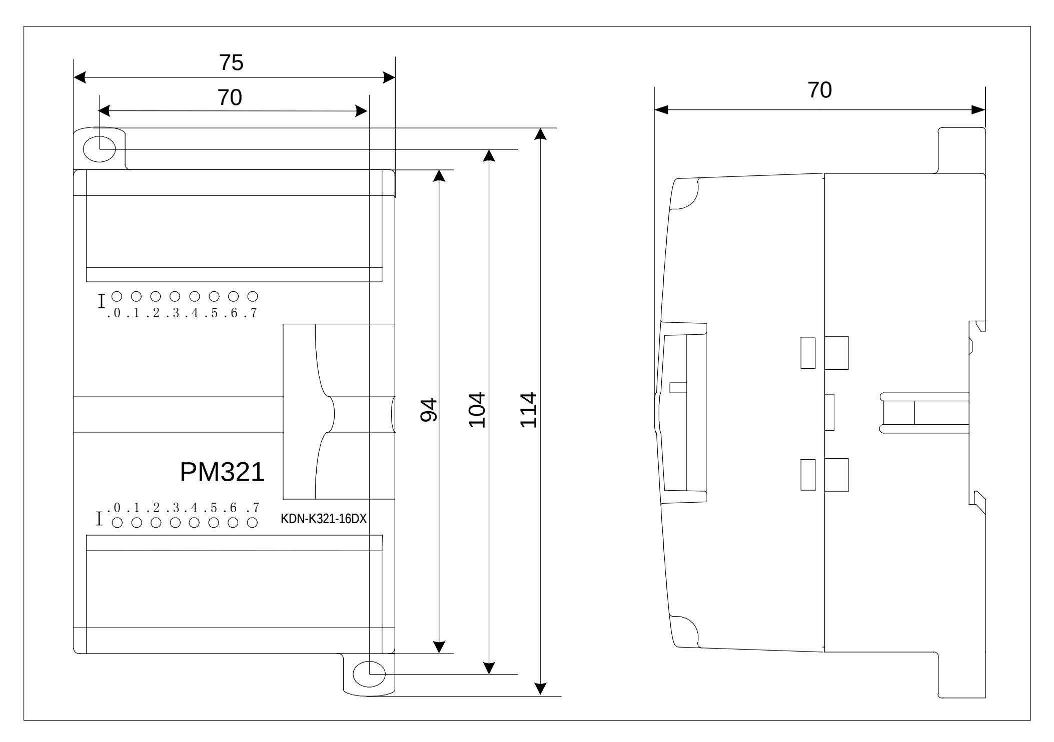


图 3-8 KDN-K321-16DX 安装尺寸图

3.2.5　技术数据

电气参数		
输入通道数	16（8 通道/组）	
输入类型	源型/漏型	
额定输入电压	DC 24V（DC15～30V 时为“1”）	
额定输入电流	4.1mA@24VDC	
逻辑“0”最大输入电压	5V@0.7mA	
逻辑“1”最小输入电压	15V@2.5mA	
输入滤波延迟	5ms	
扩展总线电流损耗	5V	104mA
	24V	-
输入与内部逻辑电路的隔离 · 隔离方式 · 隔离电压	 光电耦合器 1500VAC/1 分钟	
状态指示	绿色 LED 指示每个通道的信号“1”	
占用地址空间		
DI 映像区	2 字节	
DO 映像区	-	
尺寸和重量		
尺寸(长×宽×高)	114×75×70mm	
净重	150g	

第四章 DO 扩展模块

本章详细描述了 KDN-K3 系列 PLC 中的 DO 类扩展模块。每种模块均用独立一节详细介绍硬件原理、接线图、技术参数等信息。

DO 类扩展模块类统一称为 PM322。

4.1 DO 8×DC24V

该模块的订货号是：KDN-K322-08DT。

这是具有 8 个通道的晶体管型开关量输出模块，基本功能是接收来自于扩展总线的控制数据并经过隔离、调理放大后转换成现场所需的 DC24V 信号输出。

该模块的各个通道均有指示其输出状态的指示灯。

4.1.1 主要特点

- 8 个晶体管型输出通道，共分成 2 组，每组 4 通道；
- 额定供电电源电压 DC24V；
- 额定输出电压 DC24V，每通道最大输出电流 750mA，源型；
- 供电电源接入保护；
- 感性负载输出保护；
- 短路保护（当输出电流大于 3A 时保护）；
- 通道并联；
- 输出与内部逻辑电路之间光电隔离。

4.1.2 前面板示意图

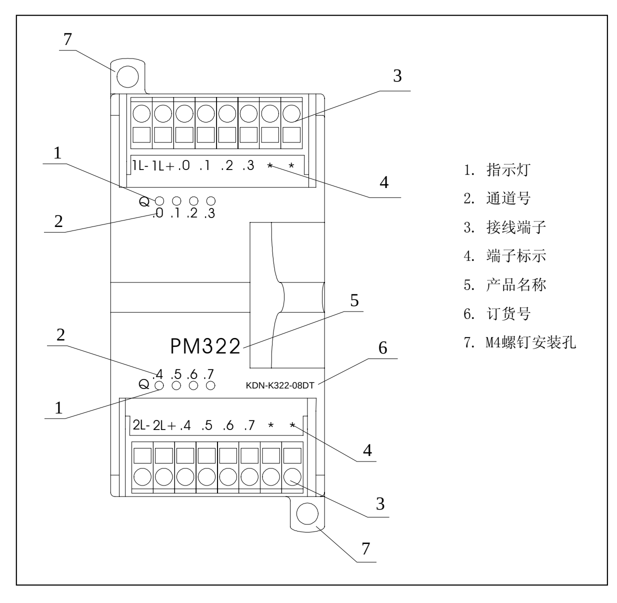


图 4-1 KDN-K322-08DT 前面板图

4.1.3 接线图和电气原理图

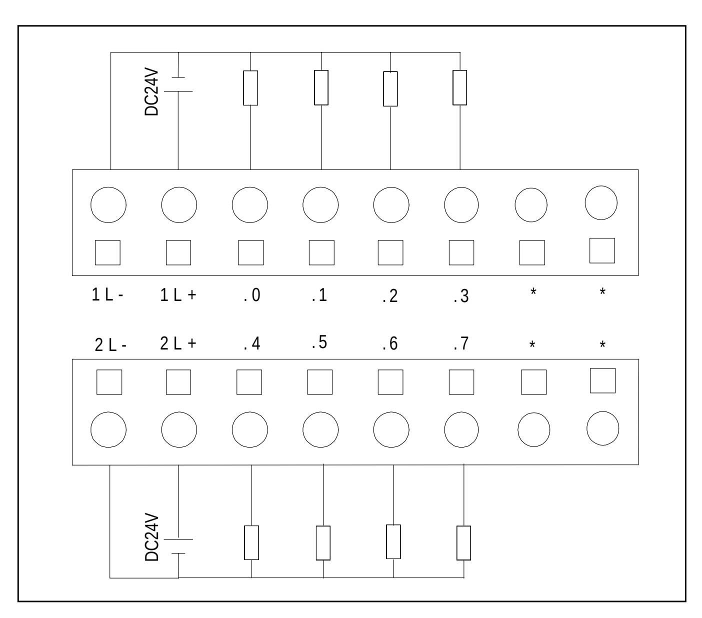


图 4-2 KDN-K322-08DT 接线图

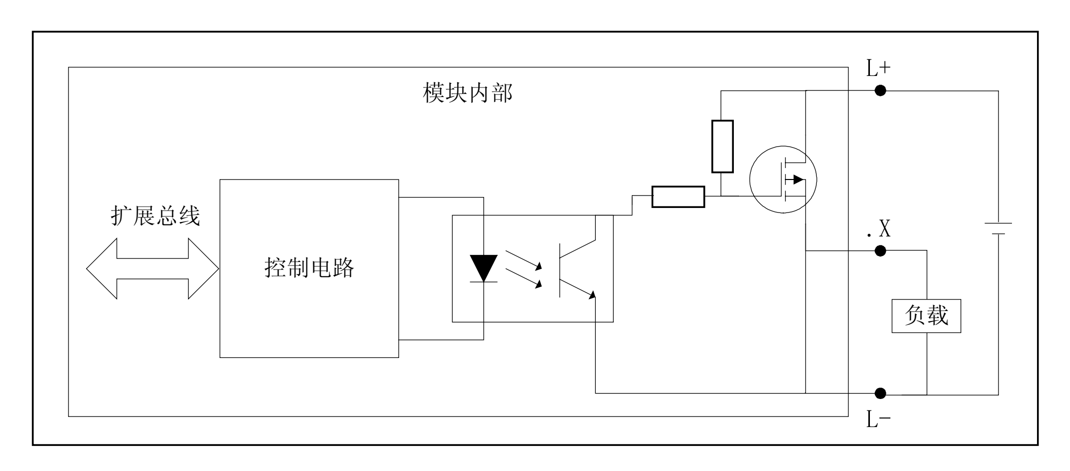


图 4-3　KDN-K322-08DT 电气原理图

4.1.4　安装尺寸图

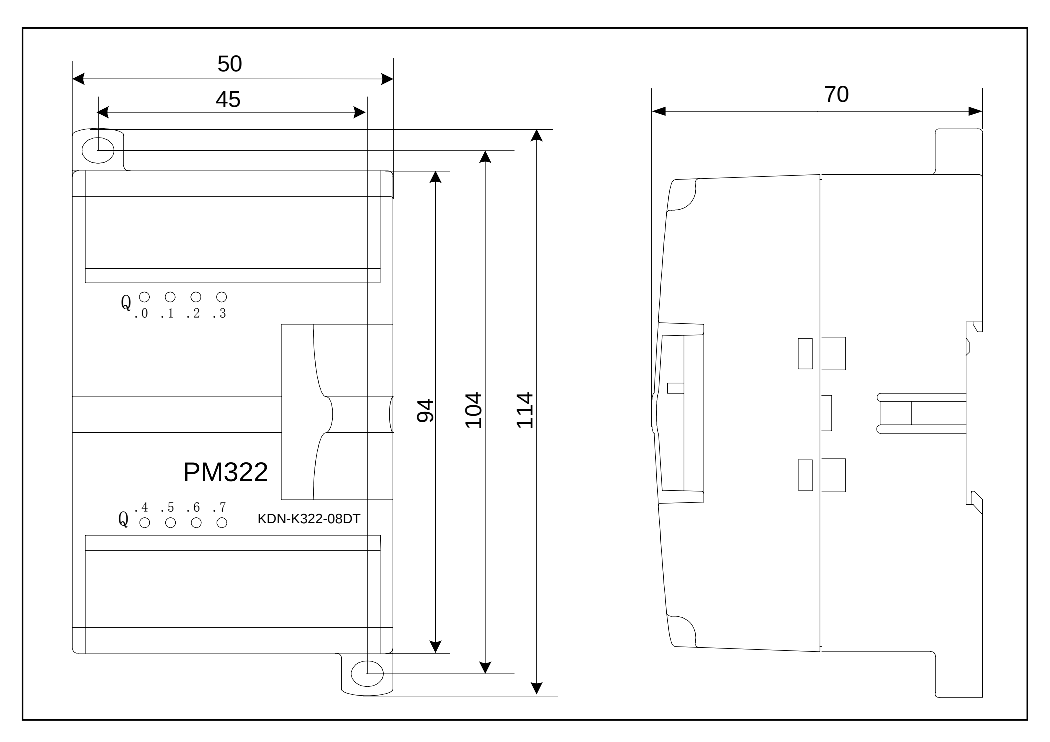


图 4-4　KDN-K322-08DT 安装尺寸图

4.1.5　技术数据

电气参数		
输出通道数	8（4 通道/组）	
输出类型	源型	
额定供电电源电压	DC 24V	
额定输出电压	DC 24V	
最大输出电流	750mA@24VDC	
输出漏电流	最大 0.5µA	
输出阻抗	最大 0.2Ω	
输出延迟时间 · 接通延时 · 断开延时	 0.3--5µs 5µs	
扩展总线电流损耗	5V	74.4mA
	24V	-
输出与内部逻辑电路的隔离 · 隔离方式 · 隔离电压	 光电耦合器 1500VAC/1 分钟	
感性负载输出保护功能	有	
短路保护功能	有（当每组输出电流大于 3A 时保护）	
通道并联功能	有	
状态指示	绿色 LED 指示每个通道的信号“1”	
占用地址空间		
DI 映像区	-	
DO 映像区	1 字节	
尺寸和重量		
尺寸(长×宽×高)	114×50×70mm	
净重	125g	

4.2 DO 8×继电器

该模块的订货号是：KDN-K322-08XR。

这是具有 8 个通道的继电器输出模块，基本功能是接收来自于扩展总线的控制数据并经过隔离、调理放大后转换为通道内继电器线圈的控制信号。

该模块的各个通道均有指示其输出状态的指示灯。

4.2.1 主要特点

- 8 个继电器型输出通道，共分成 2 组，每组 4 通道；
- 供电电源电压最高 DC30V/AC270V；
- 每通道最大输出电流 3A（DC30V/AC270V）；
- 每通道独立发光二极管指示；

4.2.2 前面板示意图

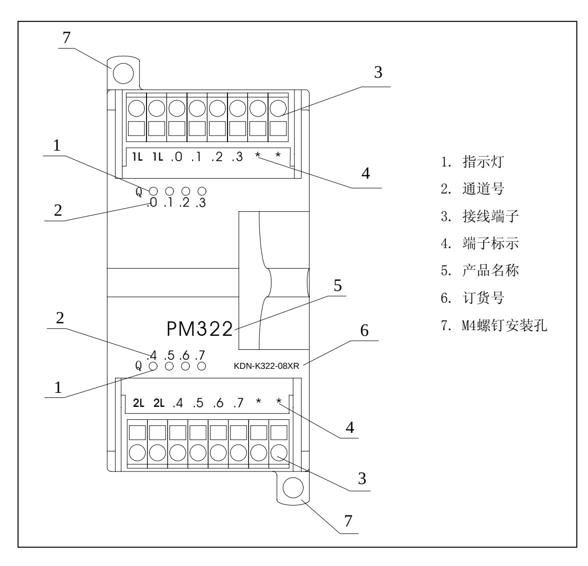


图 4-5 KDN-K322-08XR 前面板图

4.2.3　接线图和电气原理图

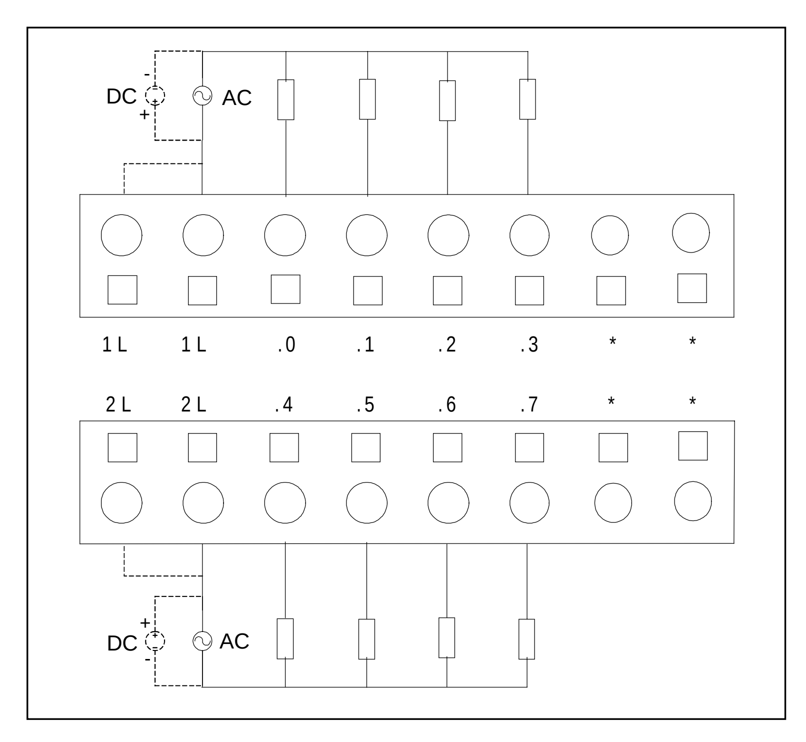


图 4-6　KDN-K322-08XR 接线图

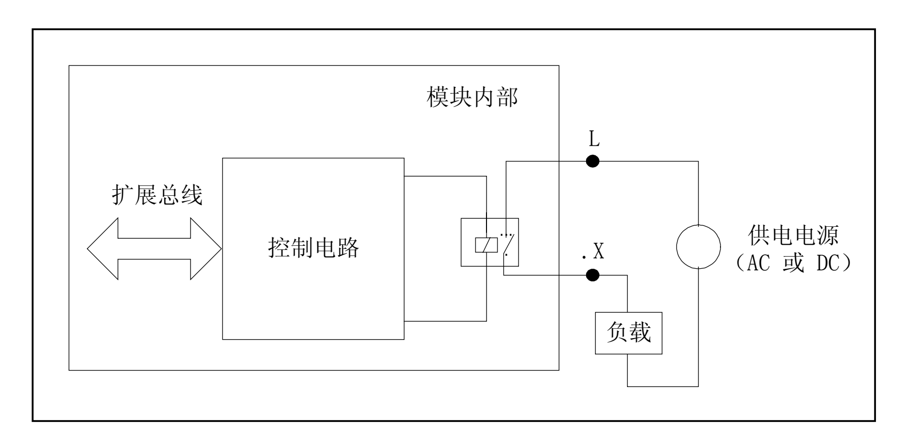


图 4-7　KDN-K322-08XR 电气原理图

4.2.4　安装尺寸图

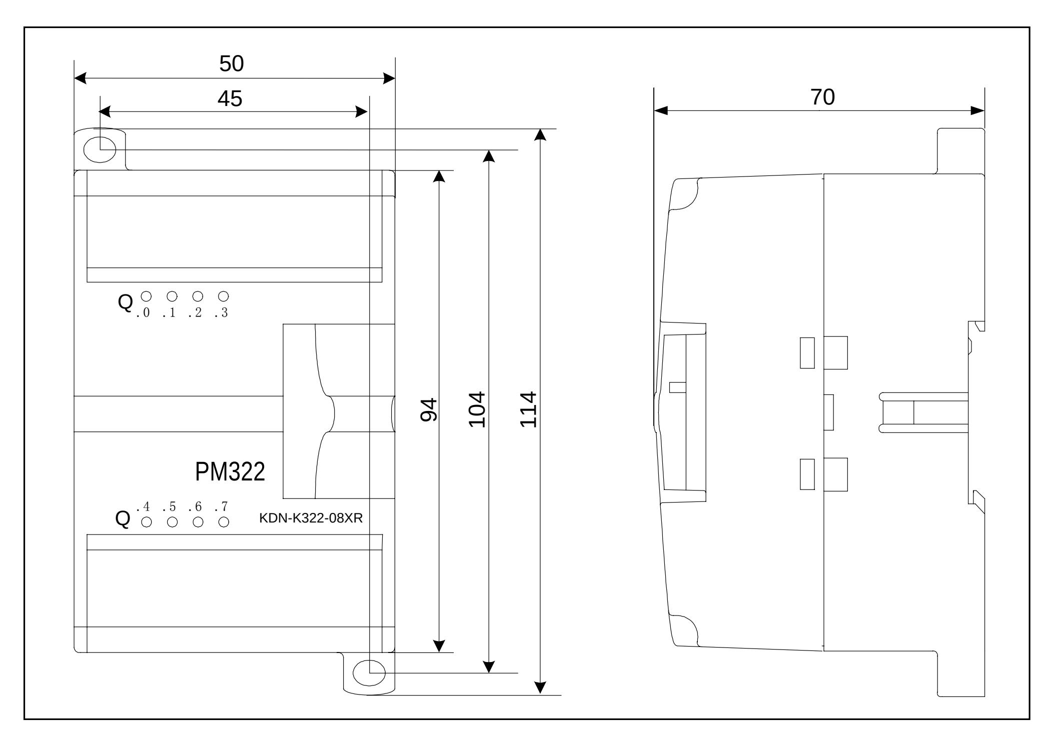


图 4-8　KDN-K322-08XR 安装尺寸图

4.2.5 技术数据

电气参数		
输出通道数	8（4 通道/组）	
供电电源电压	最高 DC 30V/AC270V	
每通道最大输出电流	3A(DC 30V/AC270V)	
每组最大输出电流	10A	
输出接通延迟时间	5ms（典型值）	
输出断开延迟时间	3ms（典型值）	
扩展总线电流损耗	5V	76.8mA
	24V	42mA
最大开关频率 · 空载 · 额定负载	 12,000 次/分钟 100 次/分钟	
触点预期寿命 · 机械寿命（空载） · 电气寿命（额定负载）	 20,000,000 次 100,000 次	
隔离特性 · 隔离方式 · 线圈与触点的隔离电压 · 触点与触点的隔离电压	 继电器 2000Vrms 750Vrms	
状态指示	绿色 LED 指示每个通道的信号“1”	
占用地址空间		
DI 映像区	-	
DO 映像区	1 字节	
尺寸和重量		
尺寸(长×宽×高)	114×50×70mm	
净重	150g	

4.3 DO 16×DC24V

该模块的订货号是：KDN-K322-16DT。

这是具有 16 个通道的晶体管型开关量输出模块，基本功能是接收来自于扩展总线的控制数据并经过隔离、调理放大后转换成现场所需的 DC24V 信号输出。

该模块的各个通道均有指示其输出状态的指示灯。

4.3.1 主要特点

- 16 个晶体管型输出通道，共分成 4 组，每组 4 通道；
- 额定供电电源电压 DC24V；
- 额定输出电压 DC24V，每通道最大输出电流 750mA，源型；
- 供电电源接入保护；
- 感性负载输出保护；
- 短路保护（当每组输出电流大于 3A 时保护）；
- 通道并联；
- 输出与内部逻辑电路之间光电隔离。

4.3.2 前面板示意图

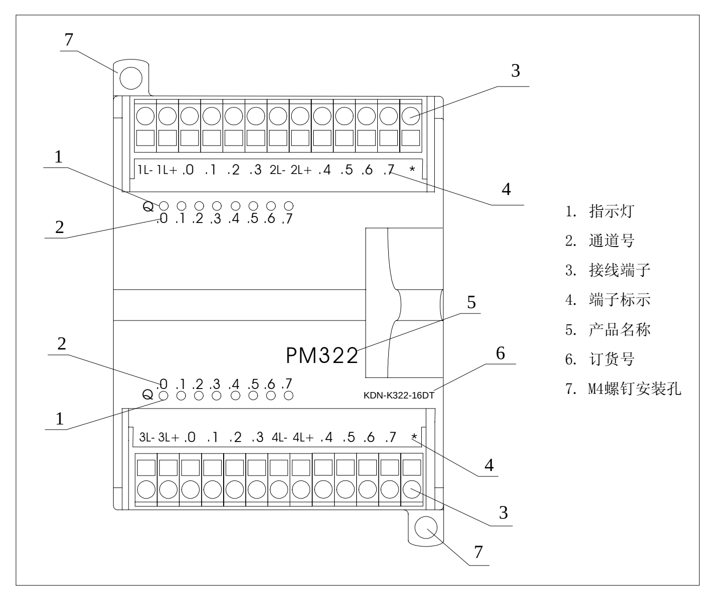


图 4-9　KDN-K322-16DT 前面板图

4.3.3　接线图和电气原理图

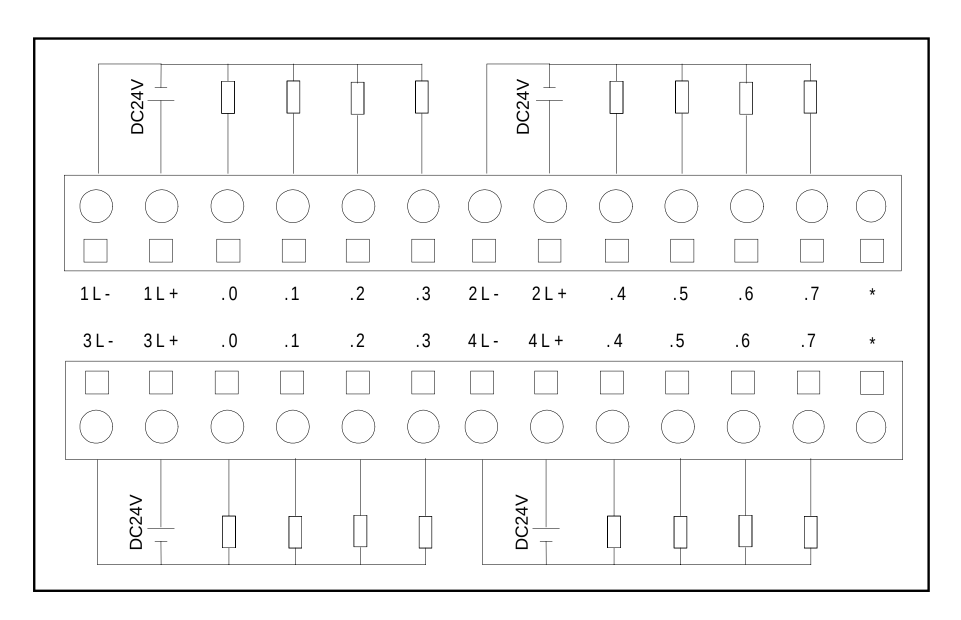


图 4-10　KDN-K322-16DT 接线图

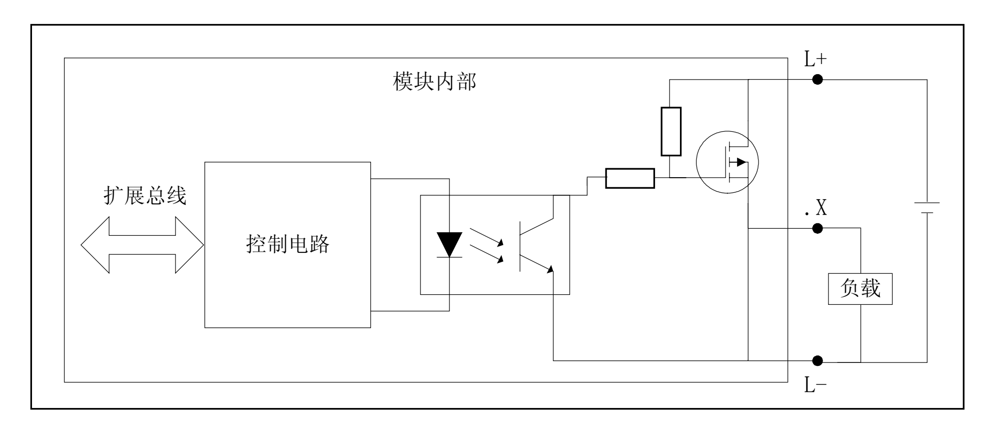


图 4-11 KDN-K322-16DT 电气原理图

4.3.4 安装尺寸图

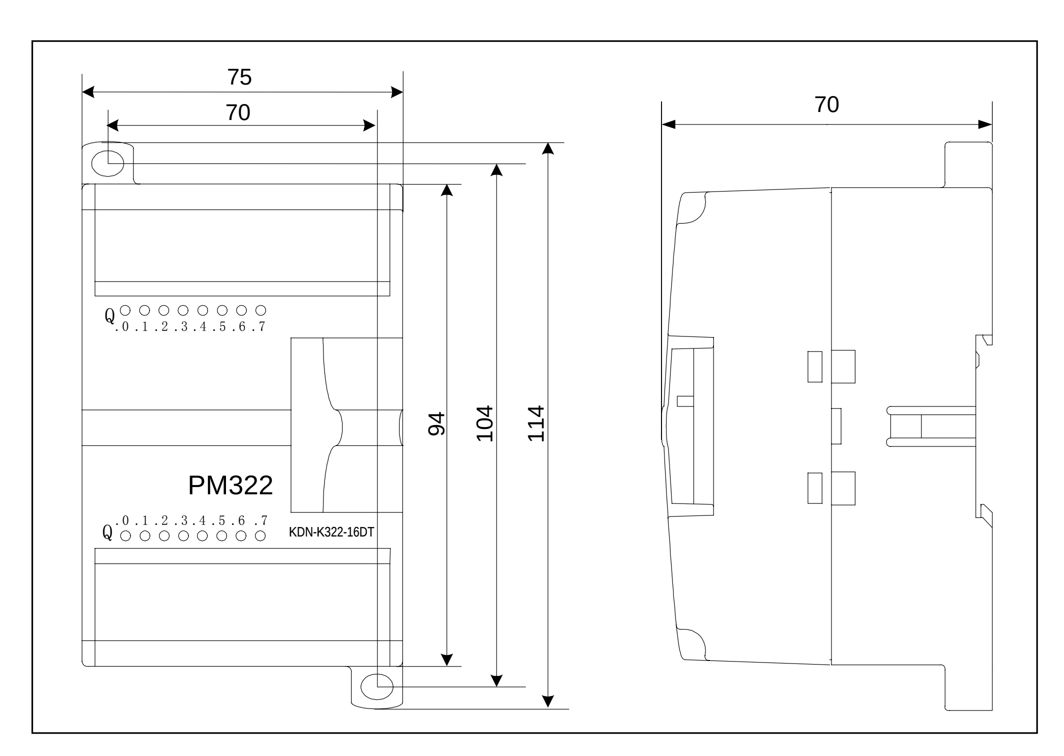


图 4-12 KDN-K322-16DT 安装尺寸图

4.3.5 技术数据

<table>
<tr><th colspan="3">电气参数</th></tr>
<tr><td>输出通道数</td><td colspan="2">16（4 通道/组）</td></tr>
<tr><td>输出类型</td><td colspan="2">源型</td></tr>
<tr><td>额定供电电源电压</td><td colspan="2">DC 24V</td></tr>
<tr><td>额定输出电压</td><td colspan="2">DC 24V</td></tr>
<tr><td>最大输出电流</td><td colspan="2">750mA@24VDC</td></tr>
<tr><td>输出漏电流</td><td colspan="2">最大 0.5μA</td></tr>
<tr><td>输出阻抗</td><td colspan="2">最大 0.2Ω</td></tr>
<tr><td>输出延迟时间
· 接通延时
· 断开延时</td><td colspan="2">
0.3--5μs
5μs</td></tr>
<tr><td rowspan="2">扩展总线电流损耗</td><td>5V</td><td>108.8mA</td></tr>
<tr><td>24V</td><td>-</td></tr>
<tr><td>输出与内部逻辑电路的隔离
· 隔离方式
· 隔离电压</td><td colspan="2">
光电耦合器
1500VAC/1 分钟</td></tr>
<tr><td>感性负载输出保护功能</td><td colspan="2">有</td></tr>
<tr><td>短路保护功能</td><td colspan="2">有（当每组输出电流大于 3A 时保护）</td></tr>
<tr><td>通道并联功能</td><td colspan="2">有</td></tr>
<tr><td>状态指示</td><td colspan="2">绿色 LED 指示每个通道的信号“1”</td></tr>
<tr><th colspan="3">占用地址空间</th></tr>
<tr><td>DI 映像区</td><td colspan="2">-</td></tr>
<tr><td>DO 映像区</td><td colspan="2">2 字节</td></tr>
<tr><th colspan="3">尺寸和重量</th></tr>
<tr><td>尺寸(长×宽×高)</td><td colspan="2">114×75×70mm</td></tr>
<tr><td>净重</td><td colspan="2">170g</td></tr>
</table>

4.4 DO 16×继电器

该模块的订货号是：KDN-K322-16XR。

这是具有 16 个通道的继电器输出模块，基本功能是接收来自于扩展总线的控制数据并经过隔离、调理放大后转换为通道内继电器线圈的控制信号。

该模块的各个通道均有指示其输出状态的指示灯。

4.4.1 主要特点

- 16 个继电器型输出通道，共分成 4 组，每组 4 通道；
- 供电电源电压最高 DC30V/AC270V；
- 每通道最大输出电流 3A（DC30V/AC270V）；
- 每通道独立发光二极管指示；

4.4.2 前面板示意图

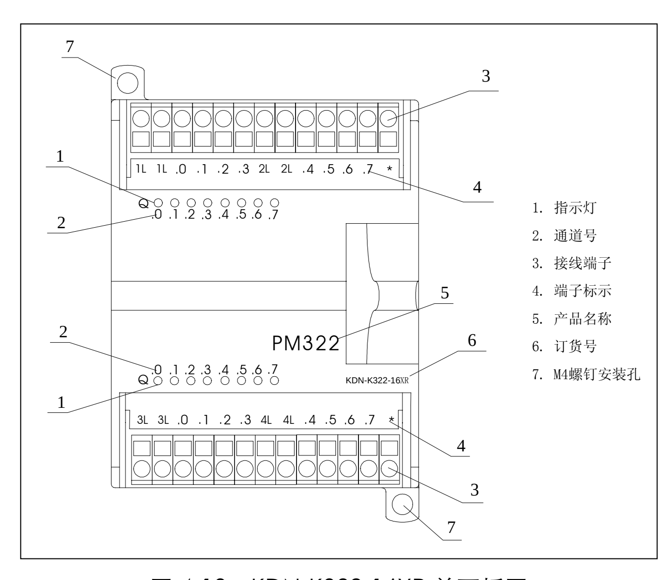


图 4-13　KDN-K322-16XR 前面板图

4.4.3　接线图和电气原理图

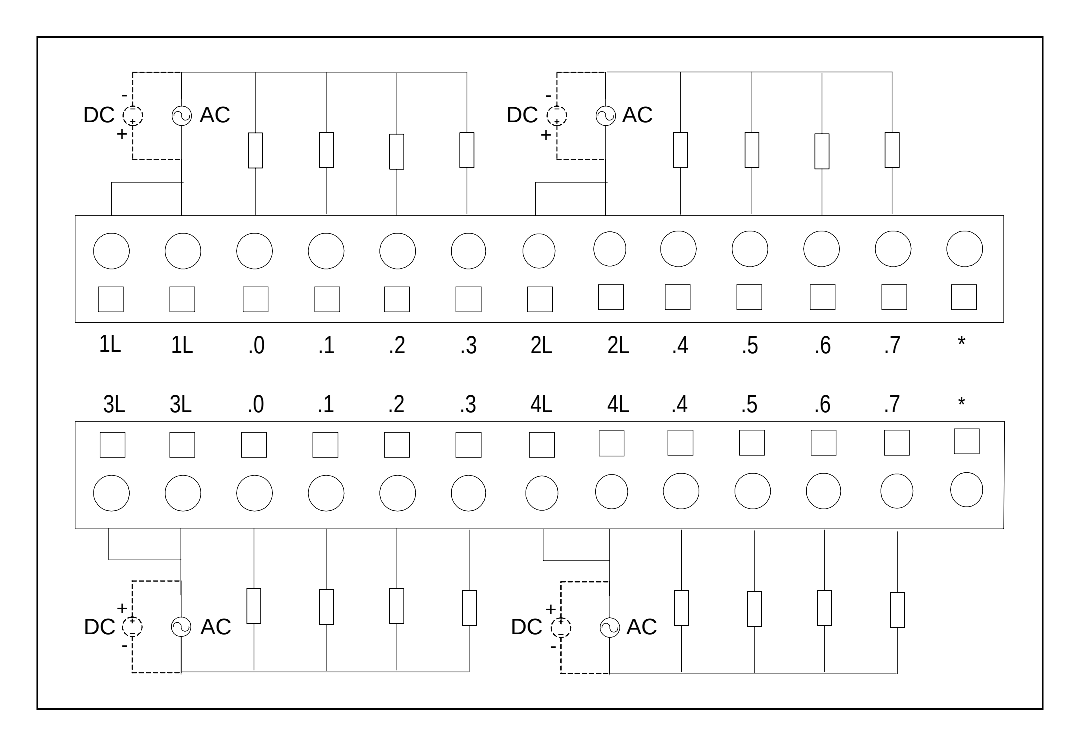


图 4-14　KDN-K322-16XR 接线图

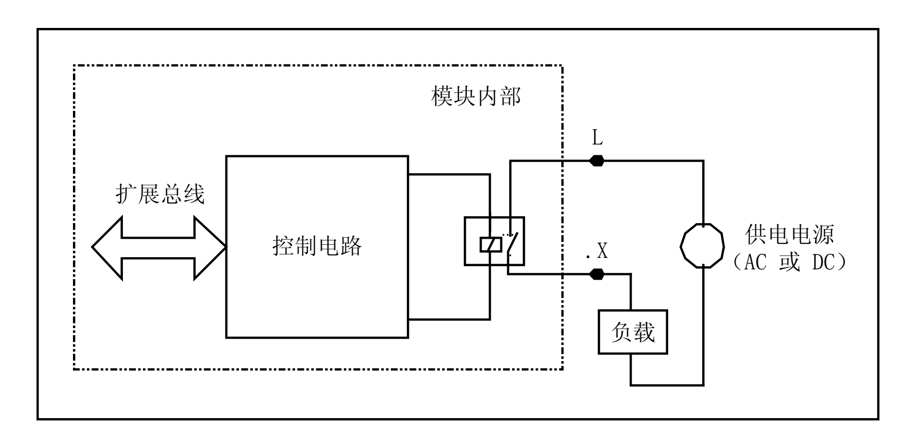


图 4-15　KDN-K322-16XR 电气原理图

4.4.4　安装尺寸图

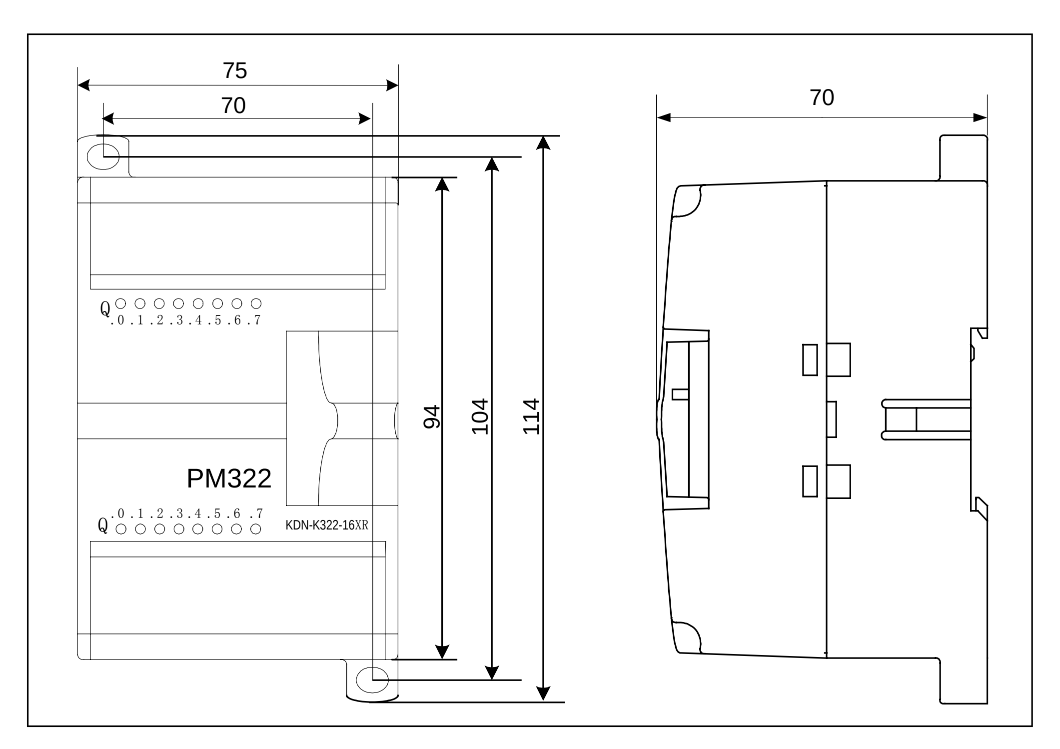


图 4-16　KDN-K322-16XR 安装尺寸图

4.4.5　技术数据

电气参数		
输出通道数	16（4 通道/组）	
供电电源电压	最高 DC 30V/AC270V	
每通道最大输出电流	3A(DC 30V/AC270V)	
每组最大输出电流	10A	
输出接通延迟时间	5ms（典型值）	
输出断开延迟时间	3ms（典型值）	
扩展总线电流损耗	5V	113.6mA
	24V	82mA
最大开关频率 · 空载 · 额定负载	 12,000 次/分钟 100 次/分钟	
触点预期寿命 · 机械寿命（空载） · 电气寿命（额定负载）	 20,000,000 次 100,000 次	
隔离特性 · 隔离方式 · 线圈与触点的隔离电压 · 触点与触点的隔离电压	 继电器 2000Vrms 750Vrms	
状态指示	绿色 LED 指示每个通道的信号“1”	
占用地址空间		
DI 映像区	-	
DO 映像区	2 字节	
尺寸和重量		
尺寸(长×宽×高)	114×75×70mm	
净重	235g	

第五章 DIO、DI/O 扩展模块

本章详细描述了 KDN-K3 系列 PLC 中的 DIO、DI/O 类扩展模块。每种模块均用独立一节详细介绍硬件原理、接线图、技术参数等信息。

在本文中，DIO 模块是指模块上的各个通道既可以作为 DI 也可以作为 DO。DI/O 模块是指模块上提供了一定数量的 DI 通道和一定数量的 DO 通道，DI 通道和 DO 通道功能唯一，不允许作它用。这两类模块统一称为 PM323。

5.1 DIO 8×DC24V

该模块的订货号是：KDN-K323-08DTX。

该模块具有 8 个通道，每个通道均可以作为 DI 或者 DO（源型），其信号形式为 DC24V。每通道在 CPU 的 DI 映像区和 DO 映像区中均占有 1 位地址，即每个通道均有两个地址：DI 地址和 DO 地址，其地址由用户在 EasyProg 软件中进行配置并下载至 CPU 中。若某通道接入的现场输入信号为 1 或者该通道的 DO 输出为 1，则该通道的 DI 值就为 1。

至于某通道是用作 DI 或者 DO，用户不需要作任何配置，只需根据实际需要接线使用即可。但在用户程序中需要注意：若某通道被实际用作 DI，则避免操作该通道的 DO 地址；同样地，若某通道被实际用作 DO，则要避免操作该通道的 DI 地址。

各个通道均有指示其输入、输出状态的指示灯。

5.1.1　主要特点

- 8 个通道，共分成 2 组，每组 4 通道，各通道均可被用作 DI 或者 DO；
- 额定供电电源电压 DC24V；
- 额定输入电压 DC24V，源型；
- 额定输出电压 DC24V，每通道最大输出电流 750mA，源型；

- 供电电源接入保护；
- 感性负载输出保护；
- 短路保护（当每组输出电流大于 3A 时保护）；
- DO 通道并联；
- 现场信号与内部逻辑电路之间光电隔离；
- 模块宽度 50mm。

5.1.2 前面板示意图

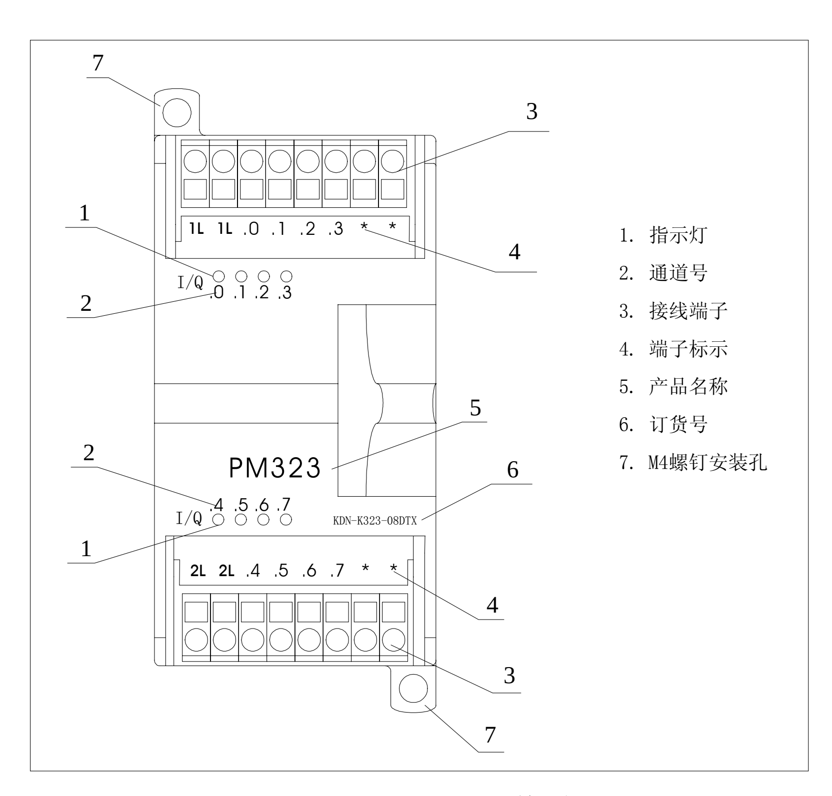


图 5-1 KDN-K323-08DTX 前面板图

5.1.3 端子接线图及电气原理图

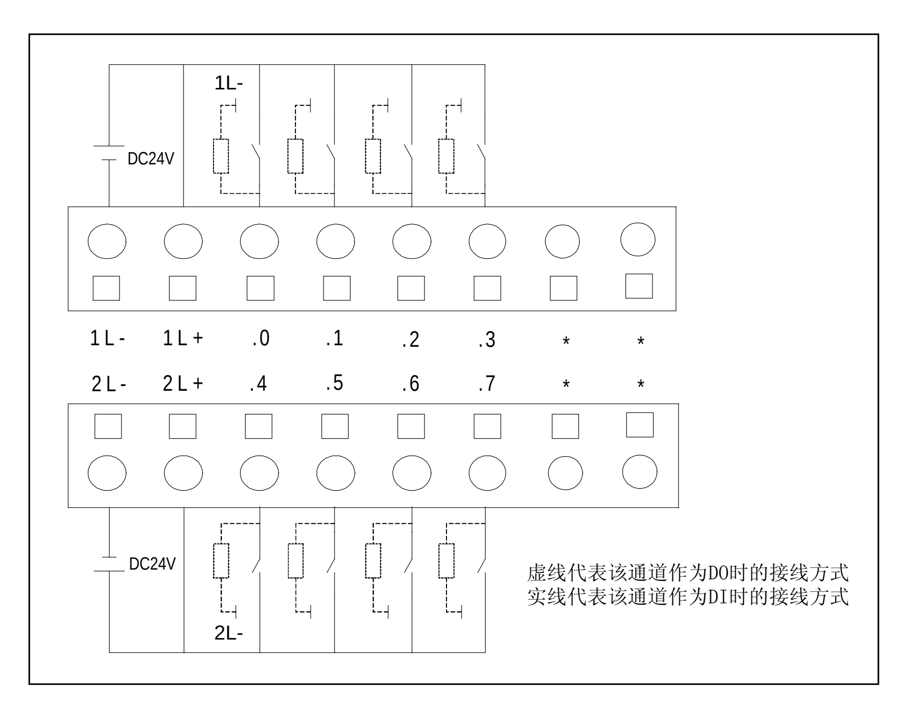


图 5-2　KDN-K323-08DTX 接线图

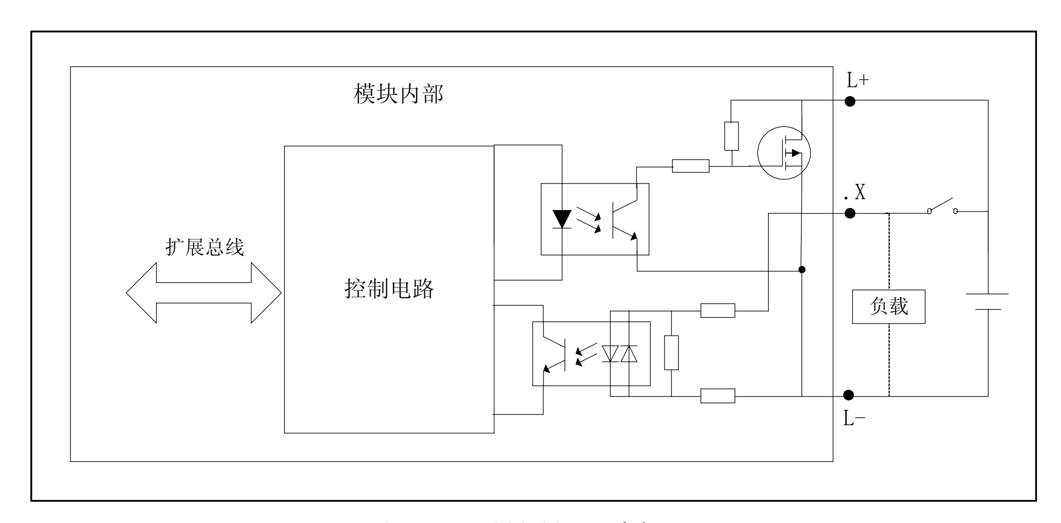


图 5-3　KDN-K323-08DTX 电气原理图

5.1.4 安装尺寸图

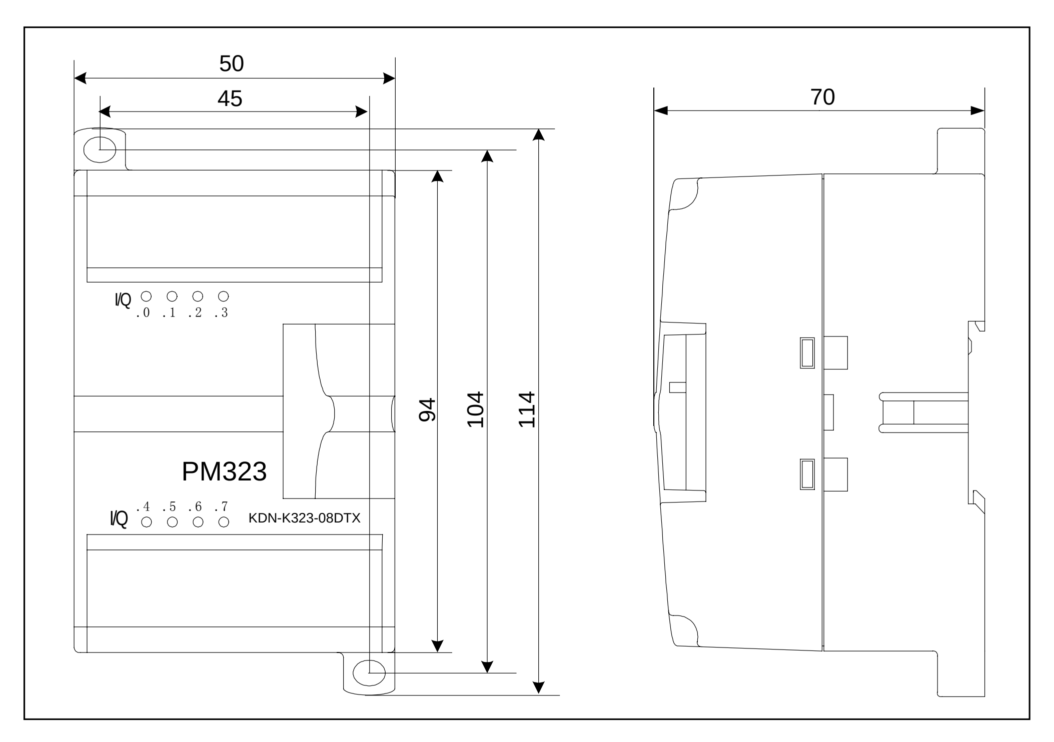


图 5-4 KDN-K323-08DTX 安装尺寸图

5.1.5　技术数据

电气参数		
通道数	8，可用作 DI 或 DO	
输入、输出类型	源型	
额定供电电源电压	DC 24V	
额定输出电压	DC 24V	
最大输出电流	750mA@24VDC	
输出阻抗	最大 0.2Ω	
输出延迟时间 · 接通延时 · 断开延时	 0.3--5μs 5μs	
额定输入电压	DC 24V（DC15～30V 时为“1”）	
额定输入电流	4.1mA@24VDC	
逻辑“0”最大输入电压	5V@0.7mA	
逻辑“1”最小输入电压	15V@2.5mA	
输入滤波延迟	5ms	
信号与内部逻辑电路的隔离 · 隔离方式 · 隔离电压	 光电耦合器 1500VAC/1 分钟	
感性负载输出保护功能	有	
短路保护功能	有（当每组输出电流大于 3A 时保护）	
通道并联功能	有	
扩展总线电流损耗	5V	106.4mA
	24V	-
状态指示	绿色 LED 指示每个通道的信号“1”	
占用地址空间		
DI 映像区	1 字节	
DO 映像区	1 字节	
尺寸和重量		
尺寸(长×宽×高)	114×50×70mm	
净重	130g	

5.2 DI/O，DI4×DC24V　DO4×DC25V

该模块的订货号是：KDN-K323-08DT。

该模块具有 8 个通道，其中 DI 4×DC24V，DO 4×DC24V，DI、DO 均为晶体管型。

各个通道均有指示其输入、输出状态的指示灯。

5.2.1　主要特点

- DI 4×DC24V，共分成 1 组；
- DI 通道既可以接源型输入（共阴极），也可以接漏型输入（共阳极）；
- DI 通道额定输入电压 DC24V，有效电压范围为 15～30V；
- DI 通道现场信号与内部电路之间光电隔离；
- DO4×DC24V，共分成 1 组，每组 4 通道；
- DO 额定供电电源电压 DC24V；
- DO 通道额定输出电压 DC24V，每通道最大输出电流 750mA，源型；
- DO 供电电源接入保护；
- DO 通道感性负载输出保护；
- DO 通道短路保护（当每组输出电流大于 3A 时保护）；
- DO 通道并联；
- DO 通道输出与内部逻辑电路之间光电隔离；
- 模块宽度 50mm。

5.2.2　前面板示意图

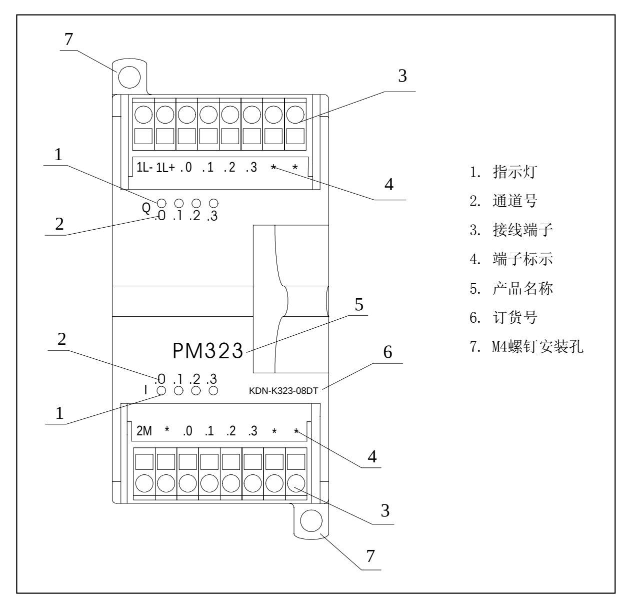


图 5-5　KDN-K323-08DT 前面板图

5.2.3　端子接线图

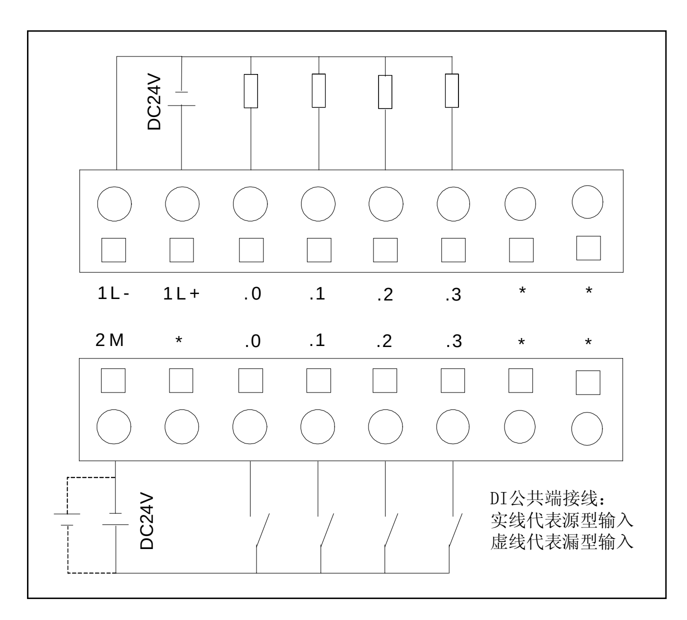


图 5-6　KDN-K323-08DT 接线图

5.2.4 安装尺寸图

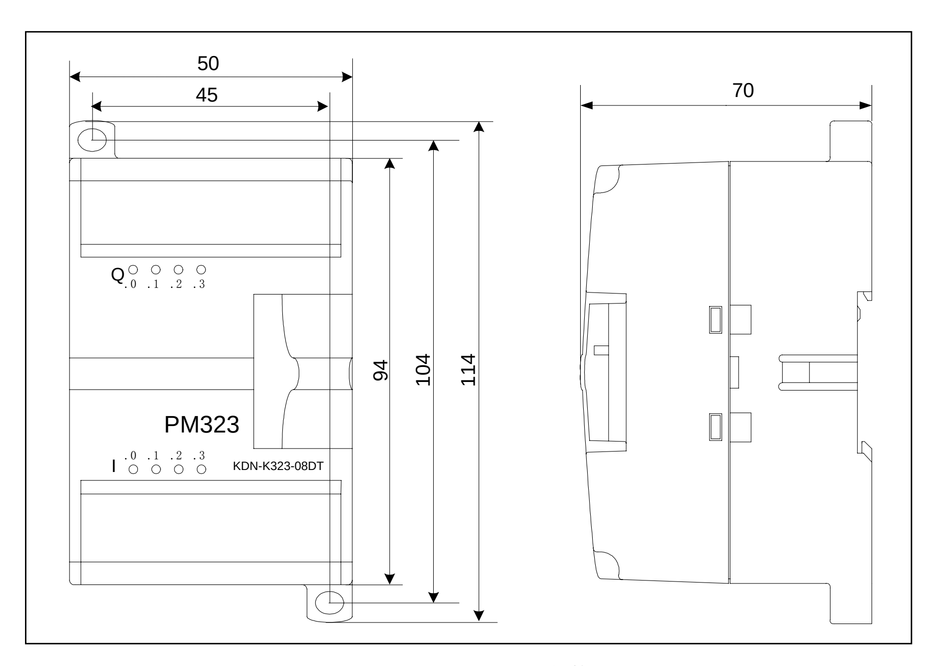


图 5-7 KDN-K323-08DT 安装尺寸图

5.2.5 技术数据

电气参数	
输入通道数	4（4 通道/组）
输入类型	源型/漏型
额定输入电压	DC 24V（DC15～30V 时为“1”）
额定输入电流	4.1mA@24VDC
逻辑“0”最大输入电压	5V@0.7mA
逻辑“1”最小输入电压	15V@2.5mA
输入滤波延迟	5ms
输入与内部逻辑电路的隔离 · 隔离方式 · 隔离电压	 光电耦合器 1500VAC/1 分钟

输出通道数	4（4 通道/组）	
输出类型	源型	
输出额定供电电源电压	DC 24V	
额定输出电压	DC 24V	
最大输出电流	750mA@24VDC	
输出漏电流	最大 0.5uA	
输出阻抗	最大 0.2Ω	
输出延迟时间 · 接通延时 · 断开延时	 0.3--5μs 5μs	
输出与内部逻辑电路的隔离 · 隔离方式 · 隔离电压	 光电耦合器 1500VAC/1 分钟	
感性负载输出保护功能	有	
短路保护功能	有（当每组输出电流大于 3A 时保护）	
通道并联功能	有	
状态指示	绿色 LED 指示每个通道的信号“1”	
扩展总线电流损耗	5V	73.2mA
	24V	-
占用地址空间		
DI 映像区	1 字节	
DO 映像区	1 字节	
尺寸和重量		
尺寸(长×宽×高)	114×50×70mm	
净重	125g	

5.3 DI/O, DI 4×DC24V　DO 4×继电器

该模块的订货号是：KDN-K323-08DR。

该模块具有 8 个通道，其中 4 个晶体管输入通道，信号形式为 DC24V，另有 4 个继电器输出通道。

各个通道均有指示其输入、输出状态的指示灯。

5.3.1　主要特点

- DI 4×DC24V，共分成 1 组；
- DI 通道既可以接源型输入（共阴极），也可以接漏型输入（共阳极）；
- DI 通道额定输入电压 DC24V，有效电压范围为 15～30V；
- DI 通道现场信号与内部电路之间光电隔离；
- DO 4×继电器，共分成 1 组；
- DO 通道供电电源电压最高 DC30V/AC270V；
- DO 每通道最大输出电流 3A（DC30V/AC270V）；
- 模块宽度 50mm。

5.3.2　前面板示意图

1. 指示灯
2. 通道号
3. 接线端子
4. 端子标示
5. 产品名称
6. 订货号
7. M4螺钉安装孔

图 5-8　KDN-K323-08DR 前面板图

5.3.3　端子接线图

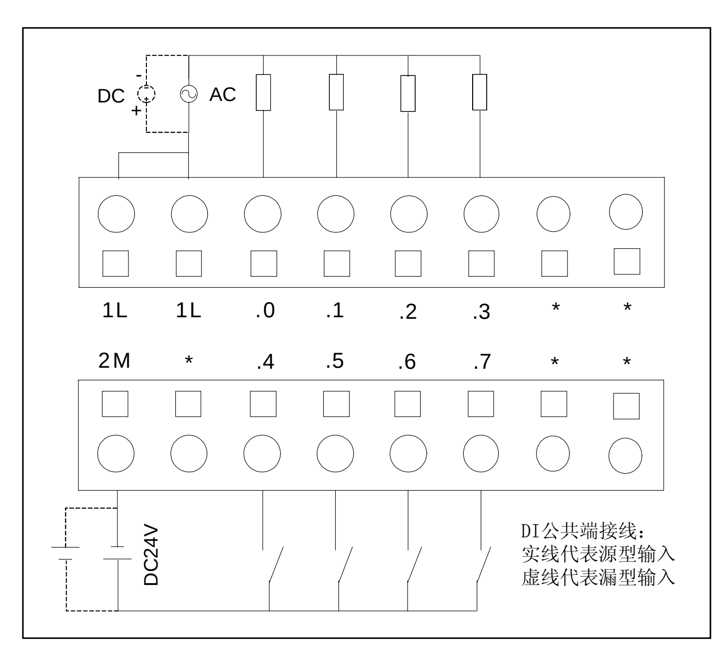


图 5-9　KDN-K323-08DR 接线图

5.3.4　安装尺寸图

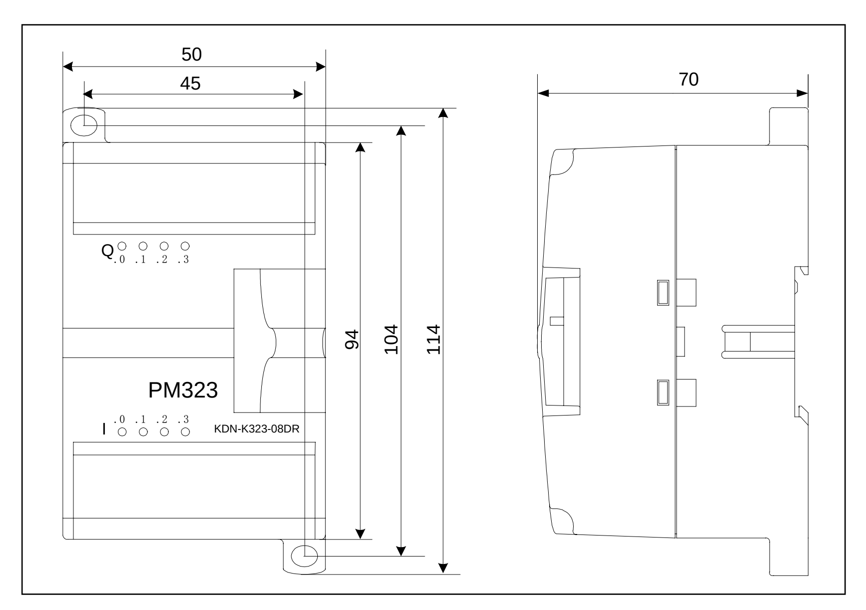


图 5-10　KDN-K323-08DR 安装尺寸图

5.3.5　技术数据

电气参数	
输入通道数	4（4 通道/组）
输入类型	源型/漏型
额定输入电压	DC 24V（DC15～30V 时为“1”）
额定输入电流	4.1mA@24VDC
逻辑“0”最大输入电压	5V@0.7mA
逻辑“1”最小输入电压	15V@2.5mA
输入滤波延迟	5ms
输入与内部逻辑电路的隔离 · 隔离方式 · 隔离电压	 光电耦合器 1500VAC/1 分钟

<table>
<tr><td>输出通道数</td><td colspan="2">4 继电器（4 通道/组）</td></tr>
<tr><td>输出供电电源电压</td><td colspan="2">最高 DC 30V/AC270V</td></tr>
<tr><td>每通道最大输出电流</td><td colspan="2">3A(DC 30V/AC270V)</td></tr>
<tr><td>每组最大输出电流</td><td colspan="2">10A</td></tr>
<tr><td>输出接通延迟时间</td><td colspan="2">5ms（典型值）</td></tr>
<tr><td>输出断开延迟时间</td><td colspan="2">3ms（典型值）</td></tr>
<tr><td>继电器最大开关频率
· 空载
· 额定负载</td><td colspan="2">
12,000 次/分钟
100 次/分钟</td></tr>
<tr><td>继电器触点预期寿命
· 机械寿命（空载）
· 电气寿命（额定负载）</td><td colspan="2">
20,000,000 次
100,000 次</td></tr>
<tr><td>输出隔离特性
· 隔离方式
· 线圈与触点的隔离电压
· 触点与触点的隔离电压</td><td colspan="2">
继电器
2000Vrms
750Vrms</td></tr>
<tr><td rowspan="2">扩展总线电流损耗</td><td>5V</td><td>74.4mA</td></tr>
<tr><td>24V</td><td>22.0mA</td></tr>
<tr><td>状态指示</td><td colspan="2">绿色 LED 指示每个通道的信号“1”</td></tr>
<tr><td colspan="3">占用地址空间</td></tr>
<tr><td>DI 映像区</td><td colspan="2">1 字节</td></tr>
<tr><td>DO 映像区</td><td colspan="2">1 字节</td></tr>
<tr><td colspan="3">尺寸和重量</td></tr>
<tr><td>尺寸(长×宽×高)</td><td colspan="2">114×50×70mm</td></tr>
<tr><td>净重</td><td colspan="2">145g</td></tr>
</table>

5.4 DI/O，DI 8×DC24V　DO 8×DC24V

该模块的订货号是：KDN-K323-16DT。

该模块具有 16 个通道，其中 DI 8*DC24V，DO 8*DC24V，DI、DO 均为晶体管型。

各个通道均有指示其输入、输出状态的指示灯。

5.4.1 主要特点

- DI 8×DC24V，共分成 1 组；
- DI 通道既可以接源型输入（共阴极），也可以接漏型输入（共阳极）；
- DI 通道额定输入电压 DC24V，有效电压范围为 15～30V；
- DI 通道现场信号与内部电路之间光电隔离；
- DO 8×DC24V，共分成 2 组，每组 4 通道；
- DO 额定供电电源电压 DC24V；
- DO 通道额定输出电压 DC24V，每通道最大输出电流 750mA，源型；
- DO 供电电源接入保护；
- DO 通道感性负载输出保护；
- DO 通道短路保护（当每组输出电流大于 3A 时保护）；
- DO 通道并联；
- DO 通道输出与内部逻辑电路之间光电隔离；
- 模块宽度 75mm。

5.4.2 前面板示意图

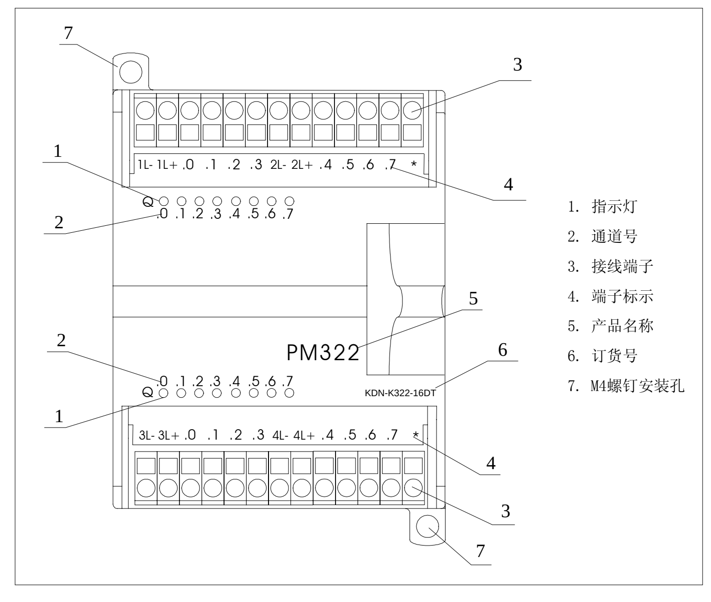


图 5-11　KDN-K323-16DT 前面板图

5.4.3　端子接线图

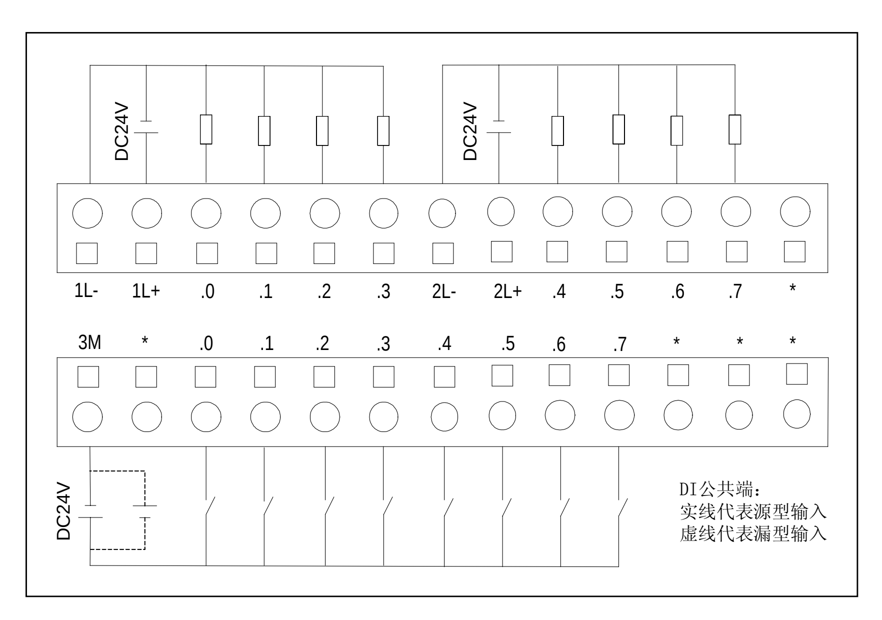


图 5-12　KDN-K323-16DT 接线图

5.4.4 安装尺寸图

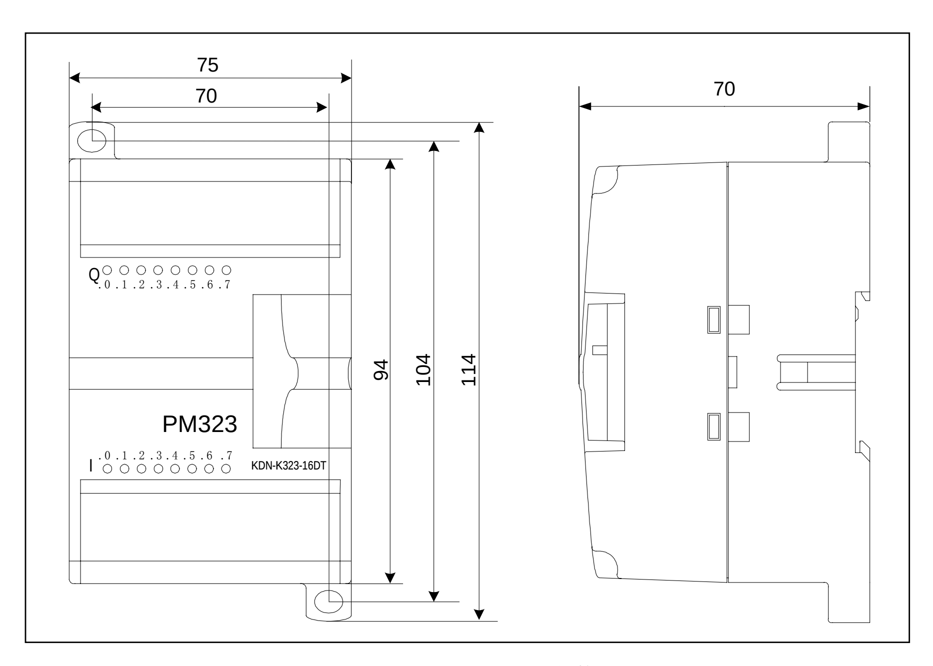


图 5-13 KDN-K323-16DT 安装尺寸图

5.4.5 技术数据

电气参数	
输入通道数	8（8 通道/组）
输入类型	源型/漏型
额定输入电压	DC 24V（DC15～30V 时为“1”）
额定输入电流	4.1mA@24VDC
逻辑“0”最大输入电压	5V@0.7mA
逻辑“1”最小输入电压	15V@2.5mA
输入滤波延迟	5ms
输入与内部逻辑电路的隔离 · 隔离方式 · 隔离电压	 光电耦合器 1500VAC/1 分钟

输出通道数	8（4 通道/组）	
输出类型	源型	
额定供电电源电压	DC 24V	
额定输出电压	DC 24V	
最大输出电流	750mA@24VDC	
输出漏电流	最大 0.5μA	
输出阻抗	最大 0.2Ω	
输出延迟时间 · 接通延时 · 断开延时	 0.3--5μs 5μs	
输出与内部逻辑电路的隔离 · 隔离方式 · 隔离电压	 光电耦合器 1500VAC/1 分钟	
感性负载输出保护功能	有	
短路保护功能	有（当每组输出电流大于 3A 时保护）	
通道并联功能	有	
状态指示	绿色 LED 指示每个通道的信号“1”	
扩展总线电流损耗	5V	106.4mA
	24V	-
占用地址空间		
DI 映像区	1 字节	
DO 映像区	1 字节	
尺寸和重量		
尺寸(长×宽×高)	114×75×70mm	
净重	160g	

5.5　DI/O, DI 8×DC24V　DO 8×继电器

该模块的订货号是：KDN-K323-16DR。

该模块具有 16 个通道，其中 8 个晶体管输入通道，信号形式为 DC24V，另有 8 个继电器输出通道。

各个通道均有指示其输入、输出状态的指示灯。

5.5.1　主要特点

- DI 8×DC24V，共分成 1 组；
- DI 通道既可以接源型输入（共阴极），也可以接漏型输入（共阳极）；
- DI 通道额定输入电压 DC24V，有效电压范围为 15～30V；
- DI 通道现场信号与内部电路之间光电隔离；
- DO 8×继电器，共分成 2 组；
- DO 通道供电电源电压最高 DC30V/AC270V；
- DO 每通道最大输出电流 3A（DC30V/AC270V）；
- 模块宽度 75mm。

5.5.2　前面板示意图

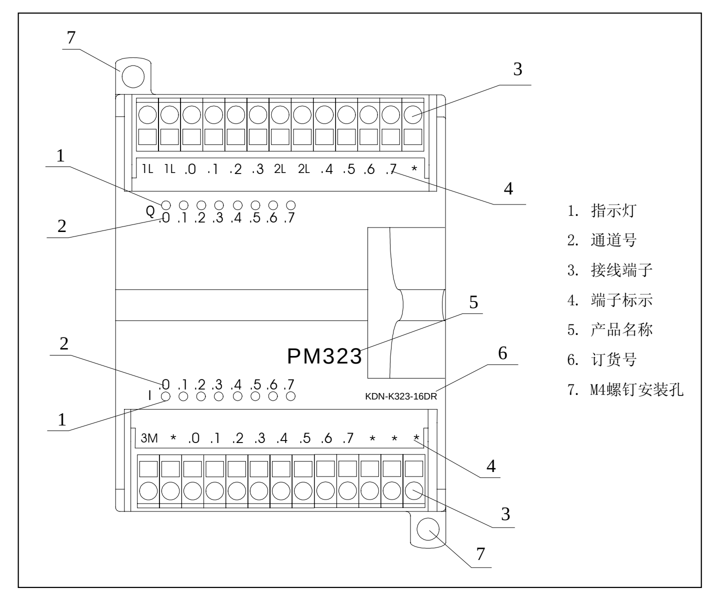


图 5-14 KDN-K323-16DR 前面板图

5.5.3 端子接线图

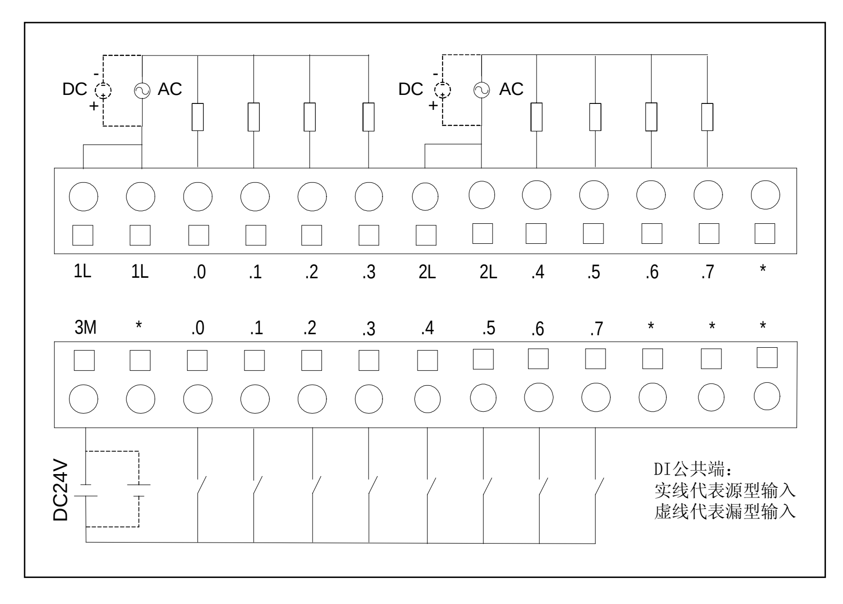


图 5-15 KDN-K323-16DR 接线图

5.5.4　安装尺寸图

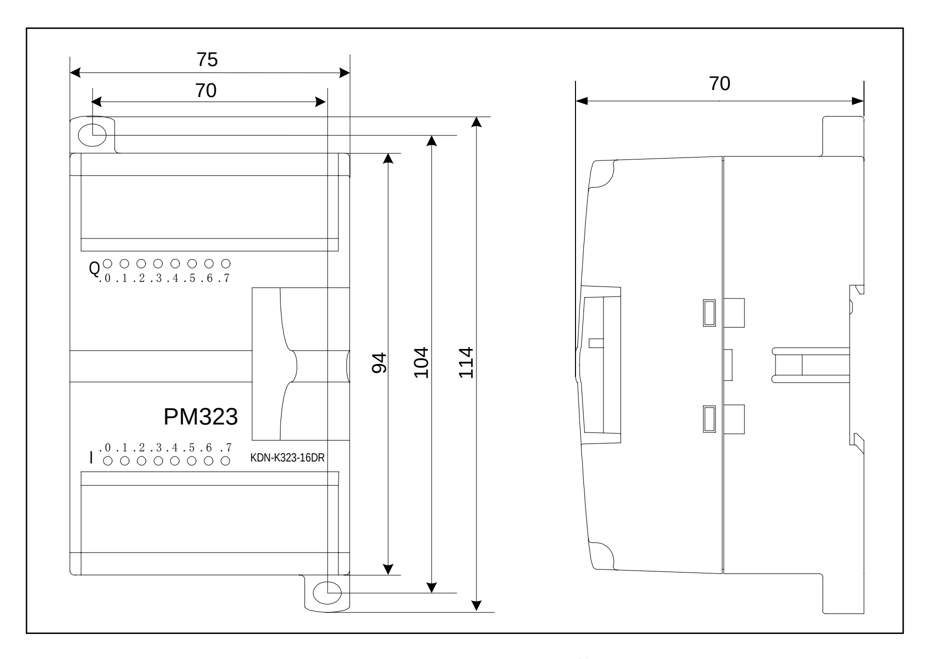


图 5-16　KDN-K323-16DR 安装尺寸图

5.5.5　技术数据

电气参数	
输入通道数	8（8 通道/组）
输入类型	源型/漏型
额定输入电压	DC 24V（DC15～30V 时为“1”）
额定输入电流	4.1mA@24VDC
逻辑“0”最大输入电压	5V@0.7mA
逻辑“1”最小输入电压	15V@2.5mA
输入滤波延迟	5ms
输入与内部逻辑电路的隔离 · 隔离方式 · 隔离电压	 光电耦合器 1500VAC/1 分钟

输出通道数	8 继电器（4 通道/组）	
输出供电电源电压	最高 DC 30V/AC270V	
每通道最大输出电流	3A(DC 30V/AC270V)	
每组最大输出电流	10A	
输出接通延迟时间	5ms（典型值）	
输出断开延迟时间	3ms（典型值）	
继电器最大开关频率 · 空载 · 额定负载	 12,000 次/分钟 100 次/分钟	
继电器触点预期寿命 · 机械寿命（空载） · 电气寿命（额定负载）	 20,000,000 次 100,000 次	
输出隔离特性 · 隔离方式 · 线圈与触点的隔离电压 · 触点与触点的隔离电压	 继电器 2000Vrms 750Vrms	
扩展总线电流损耗	5V	108.8mA
	24V	42.0mA
占用地址空间		
DI 映像区	1 字节	
DO 映像区	1 字节	
尺寸和重量		
尺寸(长×宽×高)	114×75×70mm	
净重	160g	

第六章 AI 扩展模块

本章详细描述了 KDN-K3 系列 PLC 中的 AI 扩展模块。每种模块均用独立一节详细介绍硬件原理、接线图、技术参数等信息。

这类模块统一称为 PM331。

6.1 AI 4×IV，多信号输入

该模块的订货号是：KDN-K331-04IV。

该模块具有 4 个通道，可以测量标准的电压或电流信号（4-20mA、1-5V、0-20mA、±10V），在模块中采用了 16 位的高精度 A/D 转换芯片。

该模块在 CPU 的 AI 映像区中占用 8 个字节的地址空间（每通道 2 个字节）。每个通道的参数，包括地址、信号形式、滤波方式等，均可以通过 EasyProg 软件单独进行配置，因此在一个模块中可以混合接入不同的信号并且各通道可以采用不同的滤波方式。

每个通道均有红色 LED 指示电流信号（4-20mA）的开路或者电压信号（1-5V）的短路。

6.1.1 主要特点

- 4 通道，多信号输入，可以测量 4-20mA、1-5V、0-20mA、±10V 信号；
- 各通道通过 EasyProg 软件单独进行参数配置；
- 信号测量精度 0.2% F.S.；
- 各通道电流输入不允许超过 28mA，电压输入则不允许超过±15V；
- 各通道红色 LED 指示电流信号开路或者电压信号短路；
- 模块宽度 50mm。

6.1.2 前面板示意图

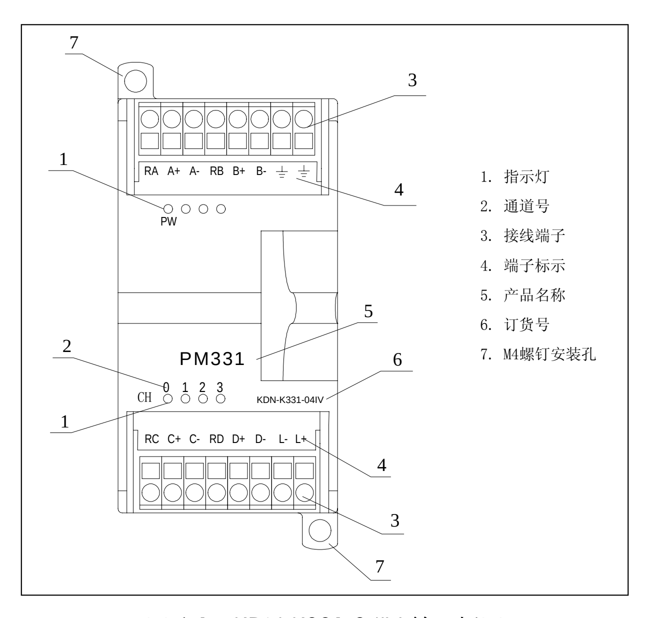


图 6-1 KDN-K331-04IV 前面板图

6.1.3 端子接线图

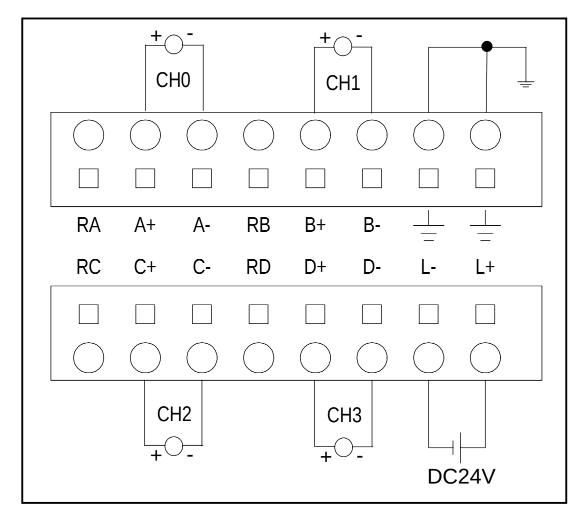


图 6-2 KDN-K331-04IV 接线图：电压信号

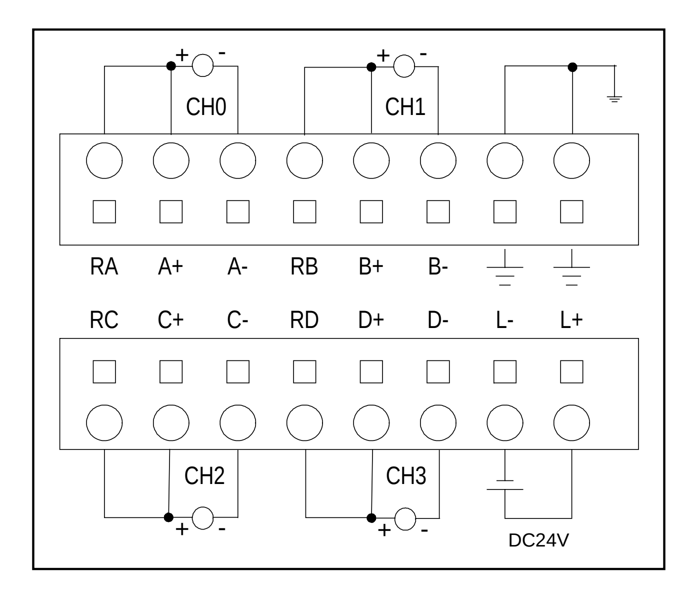


图 6-3 KDN-K331-04IV 接线图：电流信号（四线制）

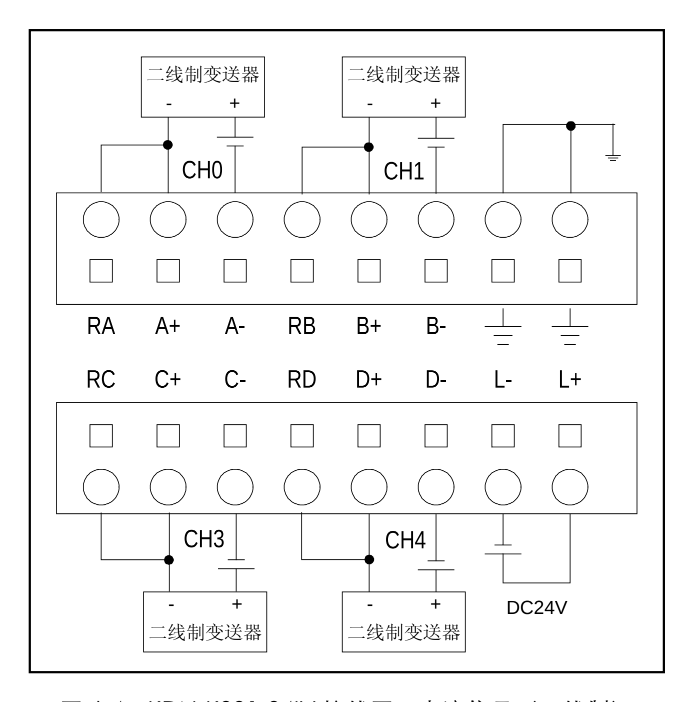


图 6-4 KDN-K331-04IV 接线图：电流信号（二线制）

6.1.4　信号测量值内部表示格式

各通道的信号首先由模块经过 A/D 转换，得到的数值再进行线性变换并将结果（在下面称之为转换值）经过扩展总线送往 CPU 模块中以供用户程序访问。各种信号能够被线性变换的范围是有限制的，若输入信号超出下限或者上限，则转换值保持下限值或者上限值不变。

下表中，I 代表输入电流值，单位 mA；V 代表输入电压值，单位 V。

信号形式	线性变换的允许输入范围	转换值
4~20mA	0.2~23.2mA	I×1000
1~5V	0.2~6V	V×1000
0~20mA	02~3.2mA	I×1000
-10~10V	-12~12V	V×1000

表 6-1　输入信号值与转换值之间的关系

6.1.5　安装尺寸图

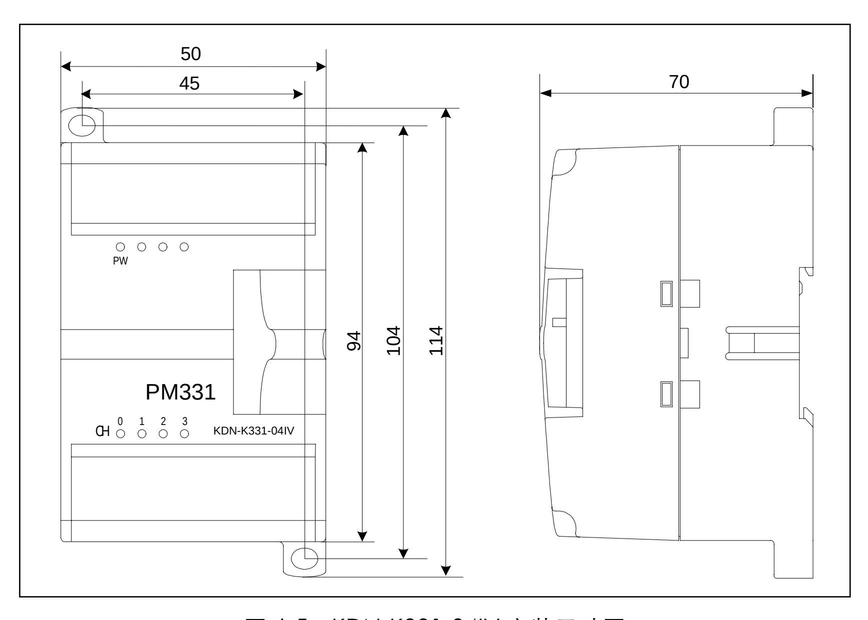


图 6-5　KDN-K331-04IV 安装尺寸图

6.1.6　技术数据

电气参数		
通道数	4	
信号形式	4~20mA、1~5V、0~20mA、±10V	
测量精度	0.2% F.S.	
输入阻抗	电流模式：250Ω 电压模式：4MΩ	
额定供电电源	DC 24V，>=75mA	
扩展总线电流损耗	5V	75mA
	24V	-
状态指示	红色 LED 指示电流开路或者电压短路	
占用地址空间		
AI 映像区	8 字节	
AO 映像区	-	
尺寸和重量		
尺寸(长×宽×高)	114×50×70mm	
净重	136g	

第七章 安装及接线

本章对 KDN-K3 系列 PLC 的安装及接线方式以及需要注意的事项进行了详细地描述。

7.1 模块外形尺寸

KDN-K3 系列 PLC 有四种尺寸规格的外壳，所有外壳的长度、高度均一样，宽度分别为 200、125、75、50mm，其中 200 和 125mm 的外壳用于 CPU 模块，75 和 50mm 的外壳用于扩展模块。合理的设计使得无论怎样组合都使系统具有优美的外观。

各种模块的详细尺寸请参阅前面各章关于模块介绍中的模块安装尺寸图。

7.2 模块的安装

7.2.1 加长扩展总线

为了使安装更加灵活，在附件中提供了加长的扩展总线电缆，加长扩展总线最大长度为 1 米。**注意：当使用扩展总线的长度超过 1 米时，建议将最后一个模块扩展接口中的第 9 针与第 10 针使用短接跳线短接起来。**

加长扩展总线的订货号如下：

KDN-K373-005	0.5 米加长总线扩展电缆
KDN-K373-010	1 米加长总线扩展电缆

7.2.2 安装方法

有两种方法将 KDN-K3 安装在控制柜内：M4 螺栓安装和 DIN 导轨卡接。安装时既可以横向排列模块，也可以纵向排列模块，甚至如果控制柜内空间零散，CPU 模块和扩展模块需要分散安装时，还可以使用加长扩展电缆进行连接。KDN-K3 的安装效果如图所示。

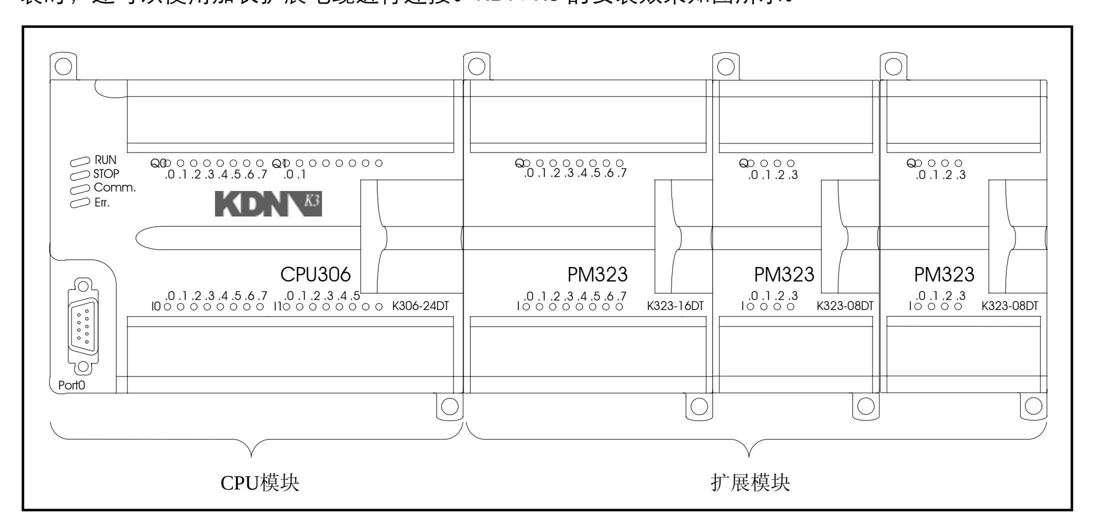


图 7-1　安装效果示意图

7.2.3　使用 M4 螺栓安装步骤

①　KDN-K3 的每一个模块都有两个 M4 螺栓安装孔，分别位于模块的右上和左下，安装时两个安装孔都要用螺栓固定。

② 首先准备安装板，根据模块的预计安装位置打好 M4 的固定孔。

③ 然后将模块从左至右或者从上至下依次用螺栓紧固在安装板上。如果是水平安装，CPU 模块应在最左侧；如果是垂直安装，CPU 模块应在最上方。在固定各扩展模块之前，先将其扩展总线插入左边或上边模块的扩展总线接口中，并调整一下让扩展总线自然滑入模块左侧的蔽线槽中以使安装后更为美观。

为防止因振动引起的松动，每套螺栓都建议加装弹簧垫圈和垫片。

7.2.4　使用 DIN 导轨安装步骤

① 准备好标准的 35mm 宽 DIN 导轨，有两种规格，如下图所示。

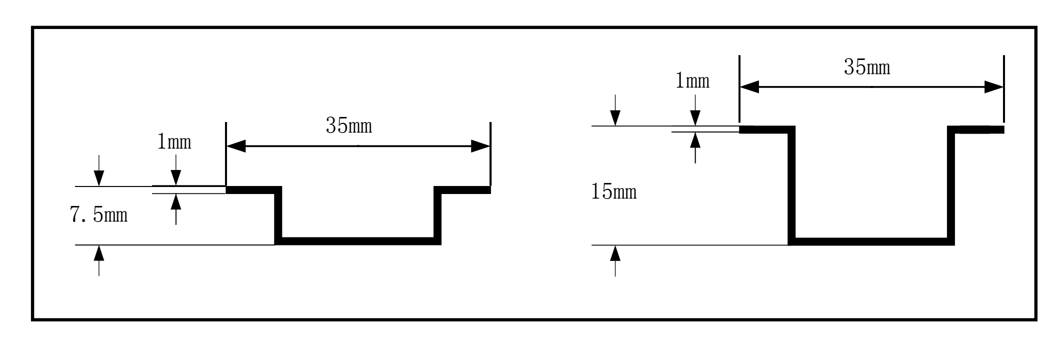


图 7-2　标准 DIN 导轨的截面规格

② 将导轨安装至需要的位置，如果水平安装，则要保证在导轨上、下方各有至少 60mm 空间；如果是垂直安装，则要保证在导轨左、右方各有至少 60mm 空间。

③ 将各个模块卡接于导轨上。方法：将模块底部的 35mm 导轨卡接滑块拉下，从导轨的上部装入模块，向前推模块下部直到模块紧贴导轨，然后再将卡接滑块推到原位即可。如下图。

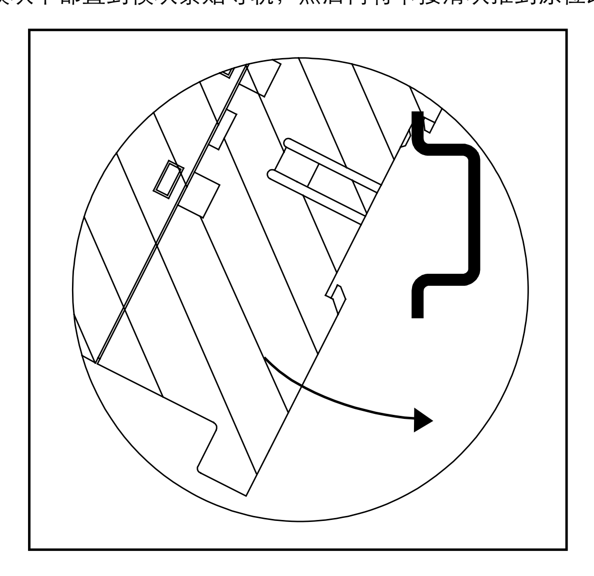


图 7-3　在 DIN 导轨上卡接模块

④ 将各扩展模块的扩展总线接入左边相邻模块的扩展总线接口中,连接完成后将全部扩展模块推紧，扩展电缆自然滑入模块左侧的蔽线槽中，使模块之间没有缝隙。

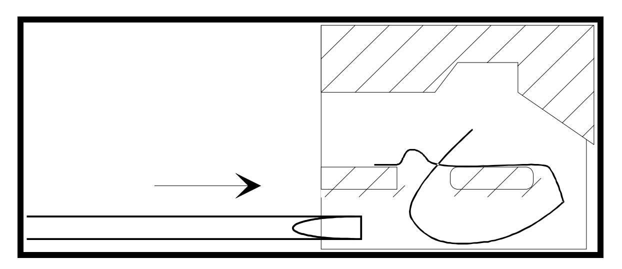
7.3 接线

7.3.1 接线端子

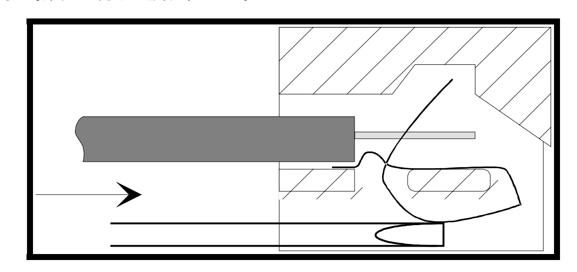
KDN-K3 系列 PLC 采用专利技术的弹簧夹持式的接线端子，提供了一种便捷、可靠的接线方式。弹簧夹持式接线有以下优点：排除人为因素，簧片自动夹紧；自锁紧机构保证线不会脱落；可以节省 75%的接线时间。

7.3.2 接线步骤

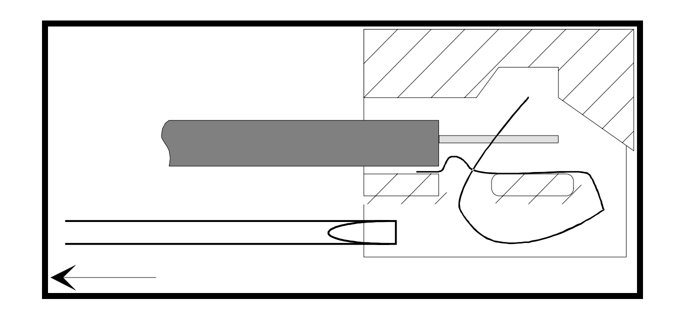
a) 将合适的螺丝刀垂直插入端子排的方孔中，压开簧片：



b) 将剥好皮的导线插入端子排的圆孔中：



c) 拔出螺丝刀。



注：① 合适的螺丝刀尺寸：

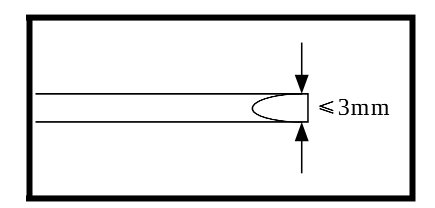


图 7-4　螺丝刀头部的宽度不能大于 3mm

② 为使簧片完全打开，螺丝刀应垂直向下插入方孔。

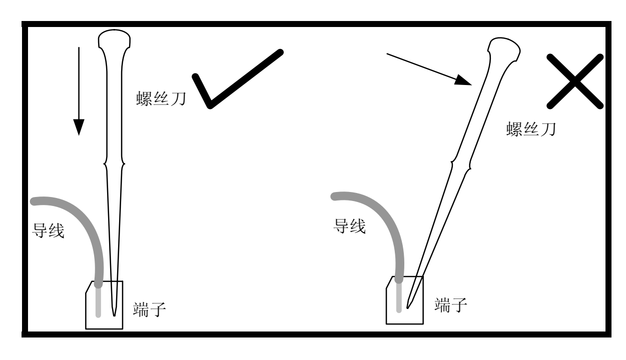


图 7-5　螺丝刀的使用方法

第三部分

EasyProg 软件手册

第一章 欢迎使用 EasyProg

EasyProg 是 KDN-K3 系列小型一体化 PLC 的上位编程软件，编程环境符合 IEC61131-3 标准，是一套功能强大、使用方便、高效的开发系统。

IEC61131-3 是 IEC 为工业自动化编程制定的标准，是在吸收不同厂家编程语言风格、方言及适应未来软件技术发展要求制定的，独立于任何一家公司，适合不同领域、不同类编程人员习惯。自发布以来得到所有顶尖 PLC 厂家的认可，各厂家的编程软件也在尽量向 IEC 标准靠拢。

EasyProg 完全是自主研发，采用了符合 IEC61131-3 标准的方案，由于许多用户通过各种渠道已经掌握了大部分编程技巧，这就使得 EasyProg 简便、易学，使用起来将会得心应手。

EasyProg 软件具有如下特点：

· 符合 IEC61131-3 标准
· 支持 IL（指令表）和 LD（梯形图）两种标准语言
· 丰富的指令集，内置 IEC61131-3 定义的标准功能、功能块以及一些特殊的应用指令
· 支持中断服务程序
· 支持用户自定义功能块（子程序）
· 允许在程序中选择显示绝对地址或者是其符号变量名称，便于工程的实施、维护
· 灵活的硬件配置方式，最大限度地允许用户自定义各种硬件参数
· 完善的联机功能，包括下载、上载、在线监测、强制、读写实时时钟等
· 定义了完善的快捷键、右键菜单，方便用户的使用
· 对用户的错误操作尽可能地予以屏蔽、提示，体贴用户的操作

第二章 EasyProg 安装及运行

本章对 EasyProg 的安装以及运行环境进行了详细的描述，可以帮助用户迈出使用 EasyProg 的第一步。若用户熟悉 Windows 下软件的使用，则可以跳过本章。

2.1 系统需求

EasyProg 软件对于计算机的要求并不高，此处做如下推荐：

2.1.1 硬件需求

- CPU：Pentium133MHz 或以上
- 硬盘：10M 以上空间
- 内存：64M 以上
- 键盘、鼠标、串行通信口（com 口）或者 USB 口（需另外使用 USB/RS232 转换器）
- 17”以上彩色显示器，分辨率设置为 1024*768

2.1.2 软件需求

Windows 9x/Me/NT4.0（sp4 以上）/2000/XP 简体中文版操作系统

2.2 安装与卸载

2.2.1 安装步骤详解

 *注意：在安装过程的任何一步单击〔取消〕，都可以终止安装。*

运行安装盘中的 EasyProgVxxxx_setup.exe（xxxx 代表版本号，比如 1004），弹出安装向导的首页，如图 2-1：



图 2-1

① 单击〔下一步〕，将确认 EasyProg 的安装路径。用户可用使用默认路径，也可以对其进行修改，如图 2-2。

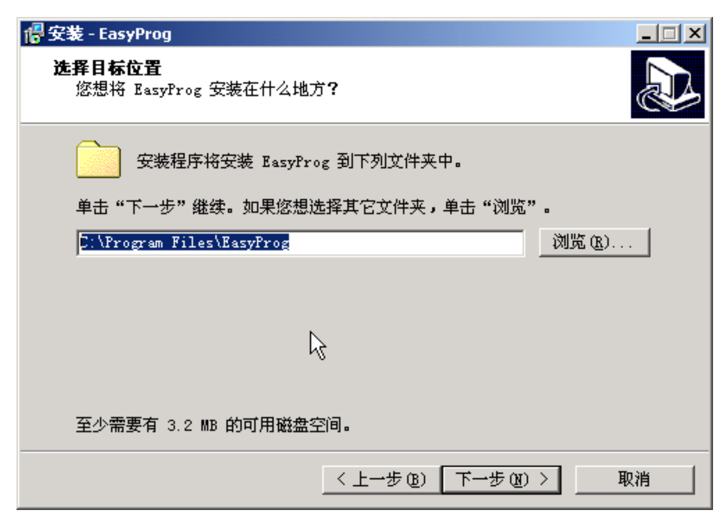


图 2-2

② 单击〔下一步〕，将确认在【开始】菜单中生成的快捷方式文件夹名称，默认为“KDN”。如图 2-3：



图 2-3

④ 单击〔下一步〕，将确认是否在桌面、运行栏中生成快捷方式。如图 **2-4**：

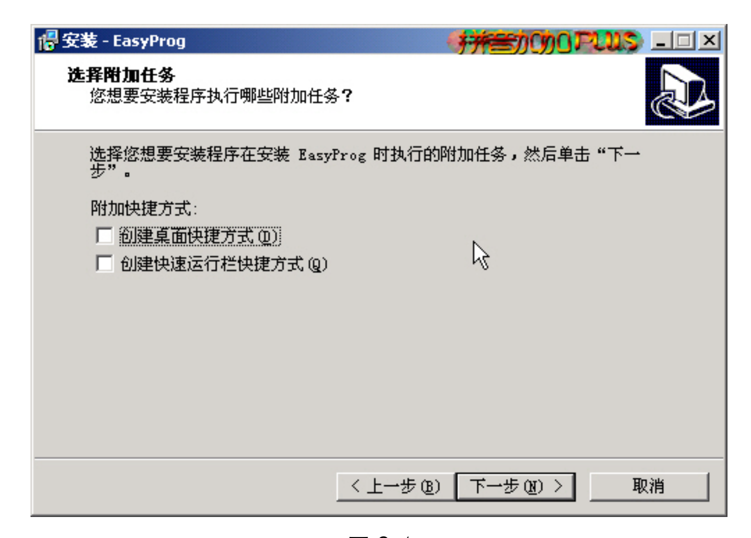


图 2-4

⑤ 用户选择完是否在桌面、运行栏中创建快捷方式后，单击〔下一步〕，将提示确认准备开始安装。如图 2-5：

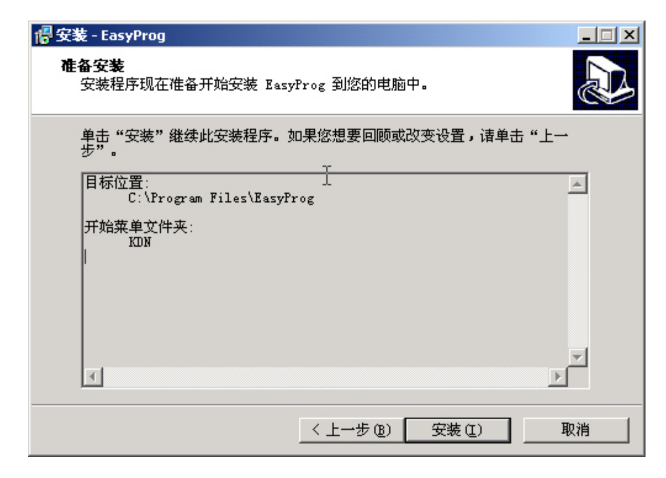


图 2-5

⑥ 单击〔安装〕，将正式开始安装过程，安装结束后的提示如图 2-6：



图 2-6

⑦ 单击〔完成〕，将结束安装过程。

用户若同时选中了〔运行 EasyProg〕，则 EasyProg 会立即启动。

2.2.2　卸载步骤详解



注意：执行卸载前，请先退出 EasyProg 程序。

卸载 EasyProg 软件共有两种方法：

· 方法一

① 执行【开始】|【程序】中 EasyProg 快捷方式组中的【卸载 EasyProg】命令，将开始卸载过程。如图 2-7 所示。

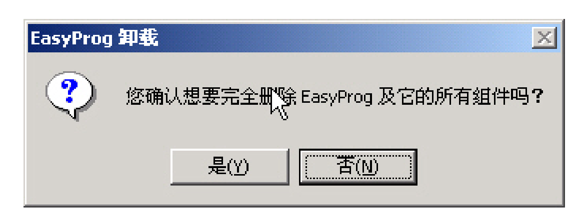


图 2-7

② 单击〔是〕，卸载程序将会干净地将 EasyProg 软件从计算机中卸载，最后提示如图 2-8。



图 2-8

· 方法二

单击【开始】|【设置】|【控制面板】，进入控制面板，执行其中的【添加或删除程序】命令，继而在弹出的“添加/删除程序”对话框中选择“EasyProg Vx.x.x.x”（x.x.x.x 代表版本号，如 1.0.0.4），然后单击〔更改/删除〕按钮，余下的操作过程如同方法一。

2.3 启动和退出 EasyProg

2.3.1　启动 EasyProg

用户可以通过下面两种简单的方法启动 EasyProg：

· 执行【开始】|【程序】中 EasyProg 快捷方式组中的【EasyProg】命令。

· 若您在安装时选中了“在桌面上创建快捷方式”，则单击桌面上的 图标即可。

2.3.2　退出 EasyProg

启动 EasyProg 后，有三种方式可以退出软件：

· 执行【文件】|【退出】菜单命令

· 使用 Alt+F4 快捷键

· 单击 EasyProg 主窗口右上角的 图标。

第三章 EasyProg 编程基础

本章详细描述了使用 EasyProg 软件针对 KDN-K3 系列 PLC 编程的基础知识，同时也介绍了IEC61131-3 标准中的一些基本概念，这些概念对于用户使用任何一种 IEC61131-3 软件都是非常有用的。本章的目的是帮助用户开始编程的学习和实践，达到“知其然并知其所以然”的程度。

在初次阅读时，并不需要用户对每个环节都理解得非常透彻，但建议用户采取“边阅读，边实验”的方式，这样会有助于对本章内容的理解。同时建议用户阅读后续章节时也采用这种方式。

3.1　程序组织单元（POU，Programme Orgnization Unit）

IEC61131-3 引入 POU 的概念，POU 就是构成项目的基本程序单元。传统的 PLC 制造商在编程方面为各自的 PLC 定义了各种类型的块，IEC61131-3 将这些块的种类减少为 3 种统一的基本类型。下面描述了标准的 POU 类型。

- 程序（Programme）

  关键字：PROGRAMME。

  “程序”用于执行一定的任务，可以有参数，无返回值。

  在所有的 POU 中，只有“程序”才能配置为“任务”在 CPU 中运行。

- 功能（Function）

  关键字：FUNCTION 。

  “功能”可以有输入参数，只有一个返回值，其返回值是通过功能名带回的。当以相同的输入参数调用功能时，其返回值总是相同的。“功能”主要用于代码重用，可被其它 POU 调用。

- 功能块（Function Block）

关键字：FUNCTION_BLOCK。

“功能块”简称 FB。可以有输入、输出参数，并具有静态变量（即能够记忆 FB 以前的状态）。FB 的输出值通过其输出参数传递。FB 的输出不仅取决于其输入值，而且也取决于在其静态变量中储存的状态值。FB 也主要用于代码重用，可被其它 POU 调用。

3.2　数据类型

数据类型定义了数据的位长度、取值范围及其初始化值。

在 IEC61131-3 中定义了一组最常用的基本数据类型，因此在 PLC 领域内这些数据类型的含义以及使用方式是开放、统一的。这些基本数据类型按照其使用目的的不同又可以划分为两类：位串型和数值型。位串型的数据，可以形象地描述为由若干个二进制位构成的串，其值可以用于逻辑运算，比如移位、按位与、按位或等；数值型的数据可以用于数学计算，比如四则运算、三角函数、求对数等。目前 K3 系列 PLC 支持的标准数据类型如表 3-1 所示。

分类	数据类型	描述	长度（位）	取值范围	缺省初始值
布尔/位串型	BOOL	布尔型	1	true, false	false
	BYTE	8 位位串	8	0 ~FF	0
	WORD	16 位位串	16	0~FFFF	0
	DWORD	32 位位串	32	0~FFFFFFFF	0
数值型	INT	整型，有符号	16	-2^{15}~（2^{15}-1）	0
	DINT	双整型，有符号	32	-2^{31} ~（2^{31}-1）	0
	REAL	实型	32	采用 ANSI/IEEE754-1985 标准；约 $1.18*10^{-38}$ ~ $3.40*10^{38}$，$-3.40*10^{38}$ ~ $-1.18*10^{-38}$	0.0

表 3-1　K3 系列 PLC 支持的数据类型

3.3 常量

在程序运行的过程中，其值不能改变的量称为常量。常量也要区分为不同的数据类型，并且根据数据类型的不同，常量的书写格式各不相同。

表 3-2 列出了 EasyProg 中各种类型常量的定义及示例。

分类	数据类型	常量格式[1]	描 述	示例
布尔/位串型	BOOL	true、alse	true 代表真，false 代表假	false
	BYTE	**B#**x	x：十进制数字，范围为 0~255	B#129
		B#2#x	x：二进制数字，范围为 0~11111111	B#2#10010110
		B#8#x	x：八进制数字，范围为 0~ 377	B#8#173
		B#16#x	x：十六进制数字，范围为 0~ FF	B#16#3E
	WORD	**W#**x	x：十进制数字，范围为 0~ 65535	W#39675
		2#x	x：二进制数字，范围为 0~ 1111111111111111	2#100110011
		W#2#x		W#2#110011
		8#x	x：八进制数字，范围为 0~ 177777	8#7432
		W#8#x		8#174732
		16#x	x：十六进制数字，范围为 0~ FFFF	16#6A7D
		W#16#x		W#16#9BFE
	DWORD	**DW#**x	x：十进制数字，范围为 0~（2^{32}-1）	DW#547321
		DW#2#x	x：二进制数字，范围为 0~11111111111111111111111111111111	DW#2#10111
		DW#8#x	x：八进制数字，范围为 0~37777777777	DW#8#76543
		DW#16#x	x：十六进制数字，范围为 0~FFFFFFFF	DW#16#FF7D
数值型	INT	**x**	x：十进制数字，范围为-32768~32767	12345
		I#x		I#-2345
		I#2#x	x：二进制数字[2]	I#2#1111110
		I#8#x	x：八进制数字[2]	I#8#16732
		I#16#x	x：十六进制数字[2]	I#16#7FFF
	DINT	**DI#**x	x：十进制数字，范围为 2^{31}~（2^{31}-1）	DI#8976540
		DI#2#x	x：二进制数字[2]	DI#2#101111
		DI#8#x	x：八进制数字[2]	DI#8#126732

		DI#16#x	x：十六进制数字(2)	DI#16#2A7FF
	REAL	带小数点的十进制数字		1.0，-243.456
		x**E**y	x：带小数点的十进制数字， 范围约 1.18~3.40，－3.40~ -1.18； y：整数，范围-38~38。	-2.3E-23

表 3-2 常量的定义



注：

(1) 在 IEC61131-3 中，标识符的使用是大小写无关的。因此在程序中将格式字符写为大写或者小写均合法，比如 W#234, dw#12345 均为合法常量。

具体请参见下文关于标识符的定义部分。

(2) INT、DINT 型常量的二进制、八进制、十六进制表示方法均采用了通用计算机中标准的补码表示法，其最高有效位（MSB）是符号位：MSB 为 1 则代表负数，MSB 为 0 则代表正数。比如 I#16#FFFF＝-1，I#7FFF＝32767，I#8000＝-32768 等。

3.4 标识符

标识符是由数字、字母等字符组成的字符串。编程人员可以使用标识符来为变量、程序等指定名称。

3.4.1 标识符的定义

标识符的定义和使用必须依据如下原则：

- 必须以一个字母或者一个单一的下划线字符开始，随后是一定数量的数字、字母或者下划线。
- 标识符是大小写无关的。比如，abc、ABC、aBC 是同一个标识符。
- 标识符的长度仅受各编程系统的限制。在 EasyProg 中，标识符的最大长度是 16 个字符。
- 关键字不允许用于用户自定义的标识符。

关键字是标准的标识符，其拼写形式和使用目的均由 IEC61131-3 明确规定。

3.4.2　标识符的使用

下面列出了在 EasyProg 中可使用标识符的语言元素：

- 程序、功能、功能块名称
- 变量
- 跳转标号等

3.5　变量

在程序的运行过程中，其值可以改变的量称为变量。变量用于初始化、存储和处理用户数据，每个变量都有其固定的数据类型。在 IEC61131-3 中，变量的存储位置可以由用户自行指定一个有效的 PLC 内存地址，也可以由编程系统自行分配。

变量的使用必须遵循“先定义，后使用”的原则。请参阅 3.4.1 标识符的定义 部分。

3.5.1　变量类型

在 IEC61131-3 中变量具有不同的形式。它们可以定义为一个 POU 的形式参数；也可以在一个 POU 内定义，仅作为本 POU 的局部变量；也可以在 POU 外定义，而且在整个工程范围内使用。下表详细描述了 IEC61131-3 中定义的各种变量类型。

变量类型	存储权限		描述
	外部	内部	
VAR	---	读写	局部变量。仅可在定义它的 POU 内使用。
VAR_INPUT	写	读	输入变量。在定义它的 POU 内作为 POU 的输入参数仅可读，在调用此 POU 时此变量仅可写。
VAR_OUTPUT	读	读写	输出变量。在定义它的 POU 内作为 POU 的输出参数可读写，在调用此 POU 时此变量仅可读。
VAR_IN_OUT	读写	读写	输入/输出变量，是 VAR_INPUT 和 VAR_OUTPUT 的组合类型。在定义它的 POU 内以及在调用此 POU 时此变量可读写。

VAR_GLOBAL	读写	读写	全局变量。可被所有的 POU 直接读写。
VAR_EXTERNAL	读写	读写	外部变量。若全局变量是在某 POU 内定义的，则若要在该 POU 外访问该变量，就必须在访问之前再次将其声明为外部变量。
VAR_ACCESS	读写	读写	关于配置（configuration）的全局变量。作为各资源之间的联系通道。

表 3-3 IEC61131-3 中的变量类型

EasyProg 目前支持 VAR、VAR_INPUT、VAR_OUTPUT、VAR_IN_OUT、VAR_GLOBAL 五种变量类型。

3.5.2 EasyProg 中变量类型的使用

在 EasyProg 中，各种变量的定义均在相应的表格中进行，这样既避免了让用户进行繁琐的输入，同时软件还能够对用户的输入进行严格的语法检查。

全局变量的定义在全局变量表中完成。

POU 的局部变量、形式参数在该 POU 编辑界面的变量定义表格中完成。

若全局变量与局部变量的名称相同，则在程序中局部变量的使用优先。

具体的使用方法请参见本手册中关于界面的详细介绍部分。

3.5.3 变量的检验

在编辑、编译程序时，EasyProg 可以自动对变量的使用情况进行检验，即判别该变量是否按照其变量类型、数据类型进行处理。这也是 IEC61131-3 的重要优点之一。

比如，在程序中为一个 WORD 类型的变量赋一个 BOOL 类型的值，或者对一个 VAR_INPUT 型的变量进行赋值操作，EasyProg 就会向用户进行错误警告并提示修改。

由于变量的特性取决于其变量类型和数据类型，因此这种检验能够在很大程度上避免由于变量使用而引起的错误。

3.6　K3 系列 PLC 内存区域分配

3.6.1 基本内存区域类型及其特性

在 KDN-K3 系列 PLC 中，CPU 的内存被划分为不同类型的几个区域，各区域的使用目的不同，并且各有自己的特性。具体如表 3-4 所示。

I	
描述	DI（开关量输入）映像区
访问方式	可按位、字节、字、双字访问
存储权限	可读
其它	允许强制，不能掉电保持
Q	
描述	DO（开关量输出）映像区
访问方式	可按位、字节、字、双字访问
存储权限	可读、可写
其它	允许强制，不能掉电保持
AI	
描述	AI（模拟量输入）映像区
访问方式	可按字（INT 型）访问
存储权限	可读
其它	允许强制，不能掉电保持
AQ	
描述	AO（模拟量输出）映像区
访问方式	可按字（INT 型）访问
存储权限	可读可写
其它	允许强制，不能掉电保持
HC	
描述	高速计数器区。用于存放各高速计数器的当前计数值。
访问方式	可按双字（DINT 型）访问
存储权限	可读
其它	不允许强制，不能掉电保持

V	
描述	变量存储区。该区域比较大，可用于存储大量的数据。
访问方式	可按位、字节、字、双字访问
存储权限	可读可写，不能掉电保持
其它	允许强制，允许掉电保持
M	
描述	中间继电器区。用于存储中间继电器的状态或者其它数据。 与 V 区相比，M 区的访问速度更快，更有利于位操作。
访问方式	可按位、字节、字、双字访问
存储权限	可读可写
其它	允许强制，允许掉电保持
SM	
描述	系统存储区。存储着系统当前的各种状态信息，并且用户可以利用 SM 区的某些位来选择和控制 CPU 的某些特殊功能。
访问方式	可按位、字节、字、双字访问
存储权限	可读可写
其它	不允许强制，不能掉电保持
L	
描述	局部变量区。程序中定义的所有局部变量、POU 的输入/输出参数均在 L 区内自动分配地址。**不建议用户直接操作 L 区**。
访问方式	可按位、字节、字、双字访问
存储权限	可读可写
其它	不允许强制，不能掉电保持

表 3-4　K3 系列 PLC 的内存区域分配

3.6.2 基本内存区域的直接寻址方式

K3 系列 CPU 的每个内存单元都有其明确、唯一的地址。用户可以通过直接寻址方式，即直接使用某内存单元的地址，来访问各内存区域。

3.6.2.1 直接地址表示格式

IEC61131-3 规定，所有的直接地址都必须以“%”开始。

各内存区域的直接寻址方式如下述列表。表中的 x、y 均代表十进制数字。

- I 区

位寻址	格式	**%I**x.y
	描述	x：字节地址，即在 I 区中该内存单元所在的字节的编号（地址）。 y：位地址，表示位于该字节的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%I0.0　%I0.7　%I5.6
字节寻址	格式	**%IB**x
	描述	x：字节地址，即在 I 区中该内存单元所在的字节的编号（地址）。
	数据类型	BYTE
	示例	%IB0　%IB1　%IB10
字寻址	格式	**%IW**x
	描述	x：字节地址，即在 I 区中该内存单元首字节的地址。 由于 WORD 类型的数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%IW0　%IW2　%IW12
双字寻址	格式	**%ID**x
	描述	x：字节地址，即在 I 区中该内存单元首字节的地址。 由于 DWORD 类型的数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT
	示例	%ID0　%ID4　%ID12

表 3-5　I 区地址格式

- Q 区

位寻址	格式	**%Q**x.y
	描述	x：字节地址，即在 Q 区中该内存单元所在的字节的编号（地址）。 y：位地址，表示位于该字节的第几位。范围为 0~7。

	数据类型	BOOL
	示例	%Q0.0　%Q0.7　%Q5.6
字节寻址	格式	**%QB**x
	描述	x：字节地址，即在 Q 区中该内存单元所在的字节的编号（地址）。
	数据类型	BYTE
	示例	%QB0　%QB1　%QB10
字寻址	格式	**%QW**x
	描述	x：字节地址，即在 Q 区中该内存单元首字节的地址。 由于 WORD 类型的数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%QW0　%QW2　%QW12
双字寻址	格式	**%QD**x
	描述	x：字节地址，即在 Q 区中该内存单元首字节的地址。 由于 DWORD 类型的数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT
	示例	%QD0　%QD4　%QD12

表 3-6　Q 区地址格式

• M 区

位寻址	格式	**%M**x.y
	描述	x：字节地址，即在 M 区中该内存单元所在的字节的编号（地址）。 y：位地址，表示位于该字节的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%M0.0　%M0.7　%M5.6
字节寻址	格式	**%MB**x
	描述	x：字节地址，即在 M 区中该内存单元所在的字节的编号（地址）。
	数据类型	BYTE
	示例	%MB0　%MB1　%MB10
字寻址	格式	**%MW**x
	描述	x：字节地址，即在 M 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT

	示例	%MW0 %MW2 %MW12
双字寻址	格式	**%MD**x
	描述	x：字节地址，即在 M 区中该内存单元首字节的地址。 由于双字数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT
	示例	%MD0 %MD4 %MD12

表 3-7 M 区地址格式

• V 区

位寻址	格式	**%V**x.y
	描述	x：字节地址，即在 V 区中该内存单元所在的字节的编号（地址）。 y：位地址，表示位于该字节的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%V0.0 %V0.7 %V5.6
字节寻址	格式	**%VB**x
	描述	x：字节地址，即在 V 区中该内存单元所在的字节的编号（地址）。
	数据类型	BYTE
	示例	%VB0 %VB1 %VB10
字寻址	格式	**%VW**x
	描述	x：字节地址，即在 V 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%VW0 %VW2 %VW12
双字寻址	格式	**%VD**x
	描述	x：字节地址，即在 V 区中该内存单元首字节的地址。 由于双字数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT 另外，V 区的最后 256 字节（VD3840—VD4092）可作 REAL 型。
	示例	%VD0 %VD4 %VD12

表 3-8 V 区地址格式

- SM 区

<table>
<tr><td rowspan="4">位寻址</td><td>格式</td><td>%SMx.y</td></tr>
<tr><td>描述</td><td>x：字节地址，即在 SM 区中该内存单元所在的字节的编号（地址）。
y：位地址，表示位于该字节的第几位。范围为 0~7。</td></tr>
<tr><td>数据类型</td><td>BOOL</td></tr>
<tr><td>示例</td><td>%SM0.0　%SM0.7　%SM5.6</td></tr>
<tr><td rowspan="4">字节寻址</td><td>格式</td><td>%SMBx</td></tr>
<tr><td>描述</td><td>x：字节地址，即在 SM 区中该内存单元所在的字节的编号（地址）。</td></tr>
<tr><td>数据类型</td><td>BYTE</td></tr>
<tr><td>示例</td><td>%SMB0　%SMB1　%SMB10</td></tr>
<tr><td rowspan="4">字寻址</td><td>格式</td><td>%SMWx</td></tr>
<tr><td>描述</td><td>x：字节地址，即在 SM 区中该内存单元首字节的地址。
由于字数据长度为 2 字节，因此 x 必须为偶数。</td></tr>
<tr><td>数据类型</td><td>WORD、INT</td></tr>
<tr><td>示例</td><td>%SMW0　%SMW2　%SMW12</td></tr>
<tr><td rowspan="4">双字寻址</td><td>格式</td><td>%SMDx</td></tr>
<tr><td>描述</td><td>x：字节地址，即在 SM 区中该内存单元首字节的地址。
由于双字数据长度为 4 字节，因此 x 必须为偶数。</td></tr>
<tr><td>数据类型</td><td>DWORD、DINT</td></tr>
<tr><td>示例</td><td>%SMD0　%SMD4　%SMD12</td></tr>
</table>

表 3-9　SM 区地址格式

- L 区（**注意：不建议用户使用直接地址来访问 L 区**）

<table>
<tr><td rowspan="4">位寻址</td><td>格式</td><td>%Lx.y</td></tr>
<tr><td>描述</td><td>x：字节地址，即在 L 区中该内存单元所在的字节的编号（地址）。
y：位地址，表示位于该字节的第几位。范围为 0~7。</td></tr>
<tr><td>数据类型</td><td>BOOL</td></tr>
<tr><td>示例</td><td>%L0.0　%L0.7　%L5.6</td></tr>
<tr><td rowspan="4">字节寻址</td><td>格式</td><td>%LBx</td></tr>
<tr><td>描述</td><td>x：字节地址，即在 L 区中该内存单元所在的字节的编号（地址）。</td></tr>
<tr><td>数据类型</td><td>BYTE</td></tr>
<tr><td>示例</td><td>%LB0　%LB1　%LB10</td></tr>
</table>

字寻址	格式	**%LW**x
	描述	x：字节地址，即在 L 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%LW0　%LW2　%LW12
双字寻址	格式	**%LD**x
	描述	x：字节地址，即在 L 区中该内存单元首字节的地址。 由于双字数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT、REAL
	示例	%LD0　%LD4　%LD12

表 3-10　L 区地址格式

- AI 区

字寻址	格式	**%AIW**x
	描述	x：字节地址，即在 AI 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	INT
	示例	%AIW0　%AIW2　%AIW12

表 3-11　AI 区地址格式

- AQ 区

字寻址	格式	**%AQW**x
	描述	x：字节地址，即在 AQ 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	INT
	示例	%AQW0　%AQW2　%AQW12

表 3-12　AQ 区地址格式

• HC 区

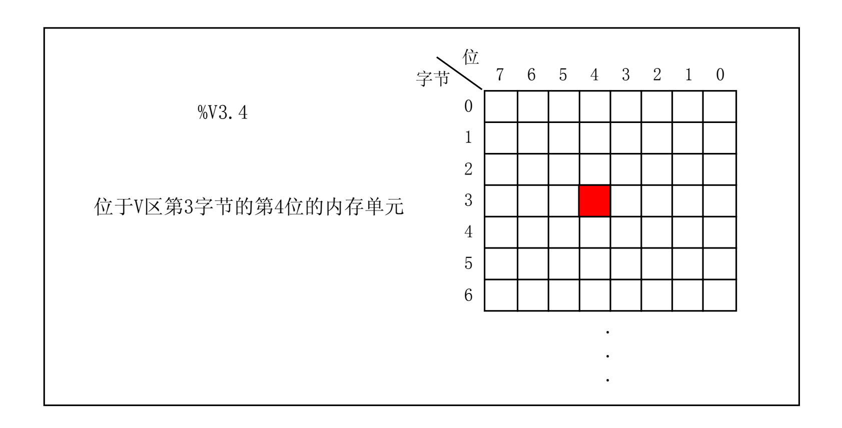
字寻址	格式	**%HC**x
	描述	x：高速计数器的编号，范围为 0~5。
	数据类型	DINT
	示例	%HC0 %HC2 %HC5

表 3-13 AQ 区地址格式

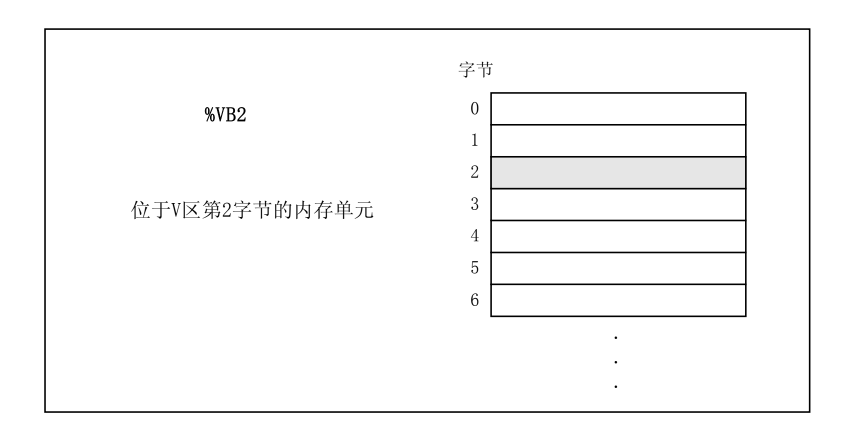
3.6.2.2 直接地址与内存单元之间的映射

每一个合法的直接地址都对应于 CPU 中的一个内存单元。在程序中对直接地址的操作就是对其对应的内存单元进行操作。下面将以 V 区为例图解直接地址与内存单元之间的映射关系。

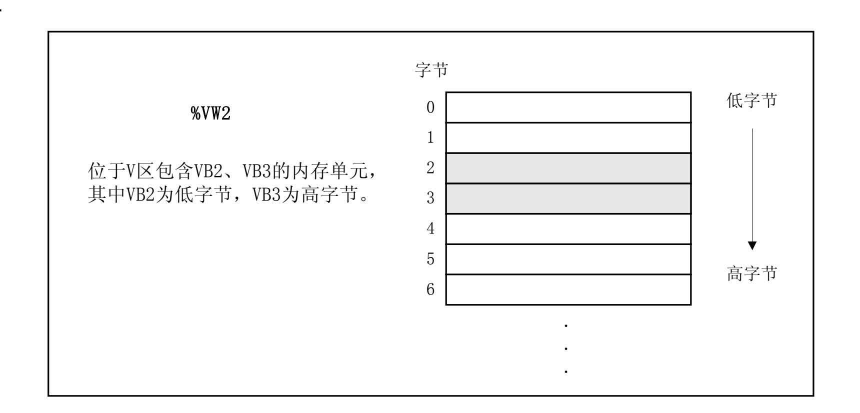
• **位地址**



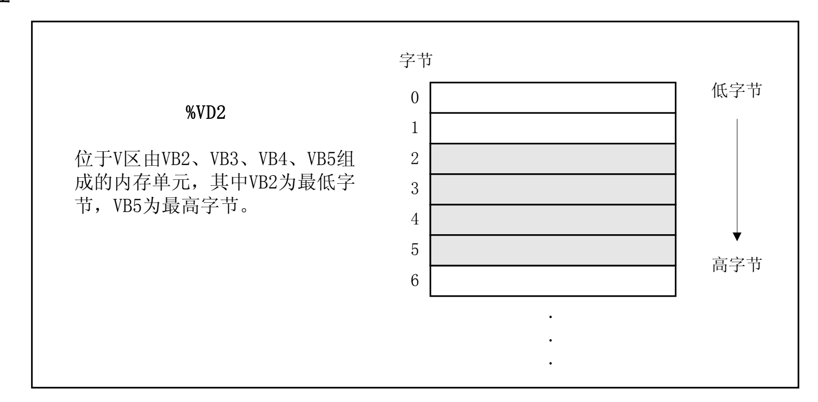
• **字节地址**


字节
%VB2
0
1
2
3
4
5
6
位于V区第2字节的内存单元

- **字地址**


字节
%VW2
0
1
2
3
4
5
6
低字节
高字节
位于V区包含VB2、VB3的内存单元，
其中VB2为低字节，VB3为高字节。

- **双字地址**


字节
%VD2
0
1
2
3
4
5
6
低字节
高字节
位于V区由VB2、VB3、VB4、VB5组
成的内存单元，其中VB2为最低字
节，VB5为最高字节。

3.6.3 各内存区域的地址范围

KDN-K3 系列 PLC 有几种不同型号的 CPU，不同型号 CPU 中的各内存区域的地址范围可能有所不同，超出允许范围的地址是非法的。表 3-14 对此进行了详细说明.

		CPU304	CPU306	CPU308
I	长度（字节）	3	8	32
	位地址	I0.0 --- I2.7	I0.0 --- I7.7	I0.0 --- I31.7
	字节地址	IB0 --- IB2	IB0 --- IB7	IB0 – IB31
	字地址	IW0	IW0 --- IW6	IW0 --- IW30
	双字地址	———	ID0 --- ID4	ID0 --- ID28
Q	长度（字节）	3	8	32
	位地址	Q0.0 --- Q2.7	Q0.0 --- Q7.7	Q0.0 --- Q31.7
	字节地址	QB0 --- QB2	QB0 --- QB7	QB0 – QB31
	字地址	QW0	QW0 --- QW6	QW0 --- QW30
	双字地址	———	QD0 --- QD4	QD0 --- QD28
AI	长度（字节）	0	32	64
	字地址	———	AIW0 --- AIW30	AIW0 --- AIW62
AQ	长度（字节）	0	32	64
	字地址	———	AQW0 --- AQW30	AQW0 --- AQW62
V	长度（字节）	4096		
	位地址	V0.0 --- V4095.7		
	字节地址	VB0 --- VB4095		
	字地址	VW0 --- VW4094		
	双字地址	VD0 --- VD4092 其中，VD3840 --- VD4092 只允许存储 REAL 型数据		
M	长度（字节）	32		
	位地址	M0.0 --- M31.7		
	字节地址	MB0 --- MB31		
	字地址	MW0 --- MW30		
	双字地址	MD0 --- MD28		
SM	长度（字节）	300		
	位地址	SM0.0 --- SM299.7		

	字节地址	SMB0 --- SMB299
	字地址	SMW0 --- SMW298
	双字地址	SMD0 --- SMD296
L	长度（字节）	16
	位地址	L0.0 --- L15.7
	字节地址	LB0 --- LB15
	字地址	LW0 --- LW14
	双字地址	LD0 --- LD12
HC	长度（字节）	24
	双字地址	HC0 --- HC5

表 3-14 K3 系列 PLC 内存区域的范围

3.6.4 FB 实例存储区的分配

3.6.4.1 IEC61131-3 中定义的标准功能块

- 定时器

  TP --- 脉冲定时器；

  TON --- 接通延时定时器；

  TOF --- 断开延时定时器。

- 计数器

  CTU --- 加计数；

  CTD --- 减计数；

  CTUD --- 加/减计数。

- 双稳态触发器

  SR --- SR 触发器；

  RS --- RS 触发器。

• 边沿检测

R_TRIG --- 上升沿检测；

F_TRIG --- 下降沿检测。

3.6.4.2　FB 的实例化

在 IEC61131-3 中，“FB 实例化”的概念特别重要。

所谓实例化，就是用户在变量定义部分通过指定变量名和数据类型来建立一个变量。

FB 也需要如同变量那样首先进行实例化。在程序中不允许直接调用 FB，而只能调用 FB 的实例。形象地讲，在程序中不能读写一个数据类型（比如 INT 型），而只能读写声明为该数据类型（比如 INT 型）的具体的变量。如下图，在程序中只能调用、访问 T1。

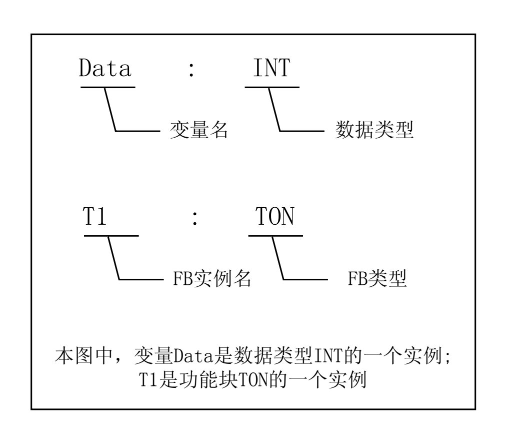


图 3-1　实例化举例

3.6.4.3　FB 实例存储区的分配

在 K3 系列 CPU 的内存内已经为每一种 FB 类型都分配了一个存储区域。当定义了某种 FB 的实例时，EasyProg 会自动在该 FB 类型对应的存储区域内为每一个实例分配一个独立的存储区。

下表描述了 K3 系列 CPU 为每种 FB 分配的实例存储区域。

T	
描述	定时器区，可在该区域内分配 TON、TOF、TP 的实例。 用于存储所有定时器实例的状态值和当前计时值。
访问方式	直接访问定时器的状态值、当前计时值
存储权限	可读
其它	允许掉电保持，不允许强制
C	
描述	计数器区，可在该区域内分配 CTU、CTD、CTUD 的实例。 用于存储所有计数器实例的状态值和当前计数值。
访问方式	直接访问定时器的状态值、计数值
存储权限	可读
其它	允许掉电保持，不允许强制
RS	
描述	RS 触发器区，可在该区域内分配 RS 的实例。 用于存储所有 RS 实例的状态值。
访问方式	直接访问 RS 状态值
存储权限	可读
其它	不允许强制，不允许掉电保持
SR	
描述	SR 触发器区，可在该区域内分配 SR 的实例。 用于存储所有 SR 实例的状态值。
访问方式	直接访问 SR 状态值
存储权限	可读
其它	不允许强制，不允许掉电保持

表 3-15　FB 示例的存储区

3.6.5　FB 实例的命名及使用

FB 的实例遵循“先定义，后使用”的原则。

为了方便用户，在 EasyProg 中特意作了如下处理：FB 实例的命名遵循传统 PLC 的常用方式，比如 T0、C3 等；用户不需要手工输入 FB 实例的定义语句，只需在程序中调用合法的功能块实例

即可，软件将会在全局变量表中为用户调用的实例自动生成定义语句。

• T

直接寻址	格式	**T**x
	描述	x：定时器编号，十进制数字。
	数据类型	BOOL --- 存储定时器的状态值 INT --- 存储定时器的当前计时值。 **T**x 兼具以上两种含义，但用户只需在程序中使用实例名即可，其含义将由软件自动识别。
	示例	T0、T5、T20

表 3-16 定时器实例

• C

直接寻址	格式	**C**x
	描述	x：计数器编号，十进制数字。
	数据类型	BOOL --- 存储计数器的状态值 INT --- 存储计数器的当前计时值。 **C**x 兼具以上两种含义，但用户只需在程序中使用实例名即可，其含义将由软件自动识别。
	示例	C0、C5、C20

表 3-17 计数器实例

• RS

直接寻址	格式	**RS**x
	描述	x：RS 触发器编号，十进制数字。
	数据类型	BOOL --- 存储 RS 触发器的状态值
	示例	RS0、RS5、RS10

表 3-18 RS 触发器实例

- SR

直接寻址	格式	**SR**x
	描述	x：SR 触发器编号，十进制数字。
	数据类型	BOOL --- 存储 SR 触发器的状态值
	示例	SR0、SR5、SR10

表 3-19 SR 触发器实例

3.6.6 FB 实例存储区的范围

系统能够为各种 FB 类型分配的存储区域的大小受到硬件本身资源的限制，因此，K3 系列各型号的 CPU 为各 FB 实例存储区分配的范围有所不同，如下表所示：

		CPU304	**CPU306**	**CPU308**
T	允许数量	32	128	256
	范围	T0 --- T31	T0 --- T127	T0 --- T255
	分辨率	待定	T0 --- T3：1ms T4 --- T19：10ms T20 --- T127：100ms	待定
	最大定时时间	32767*分辨率	32767*分辨率	32767*分辨率
C	允许数量	32	128	256
	范围	T0 --- T31	T0 --- T127	T0 --- T255
	最大计数值	32767	32767	32767
RS	最大允许数量	16		
	范围	RS0 --- RS15		
SR	最大允许数量	16		
	范围	SR0 --- SR15		

表 3-20 FB 实例存储区的分配

3.7 EasyProg 中应用程序的组织

3.7.1 工程的组织结构

在 EasyProg 中应用程序被组织成“工程（Project）”，工程中包含了用户程序、硬件配置等应用程序的所有信息。

下表详细描述了工程的组织结构。表中注明“可选”的项表示此项并非工程的必要元素，用户在工程中可以忽略它们。

<table>
<tr><td rowspan="4">程序</td><td colspan="2">初始化数据表（可选）</td><td>用户可以在此表中为 V 区中的变量指定初始值。
允许类型：BYTE、WORD、INT、DWORD、DINT、REAL。
在 CPU 上电时进入主循环之前，初始化数据表被处理一次。</td></tr>
<tr><td colspan="2">主程序</td><td>CPU 的主循环任务。在一个工程中有且仅有一个主程序。
主程序实际是 PROGRAMME 类型的 POU。</td></tr>
<tr><td colspan="2">中断服务程序（可选）</td><td>中断任务，当指定的中断事件发生时才会执行。
在一个工程中最多允许有 32 个中断服务程序
中断服务程序实际是 PROGRAMME 类型的 POU。</td></tr>
<tr><td colspan="2">子程序（可选）</td><td>可重用的程序模块。只有在主程序或者中断服务程序中被调用时才能得以执行。在一个工程中最多允许有 32 个子程序。
子程序实际是 PROGRAMME 类型的 POU。</td></tr>
<tr><td rowspan="2">配置</td><td rowspan="2">资源</td><td>硬件配置</td><td>用户在此对工程中用到的 PLC 模块及其参数进行配置。
CPU 将在冷启动时读取一次硬件配置。</td></tr>
<tr><td>全局变量表（可选）</td><td>用户在此定义工程中需要的全局变量。</td></tr>
</table>

表 3-21 工程的组织结构

3.7.2 工程的存储目录

在建立工程时 EasyProg 软件将会要求用户输入工程的存放路径，工程主文件（扩展名为.kpr）就存放于此路径下。另外，在此路径下将会自动建立一个与工程名称相同的子目录，用于存放该工程所有的程序文件、变量文件以及其它的一些临时文件等。

例如，用户若选择在 c:\temp 目录下建立一个名为 project 的工程，则工程主文件的路径是 c:\temp\project.kpr，其它文件存放在 c:\temp\project 目录中。

3.8　CPU 中程序的执行

在 CPU 中只有主程序、中断服务程序才能够被直接执行，其中主程序是被循环连续执行的，是主循环，中断服务程序在用户指定连接的中断事件发生时方能执行。

CPU 连续执行用户主程序的过程称为扫描。如图 3-2，在扫描周期内 CPU 将执行如下任务：

- 执行自诊断
- 读取物理输入通道的状态并将其写至输入映像寄存器
- 执行用户程序
- 处理通讯请求
- 将输出寄存器中的状态写至物理输出通道

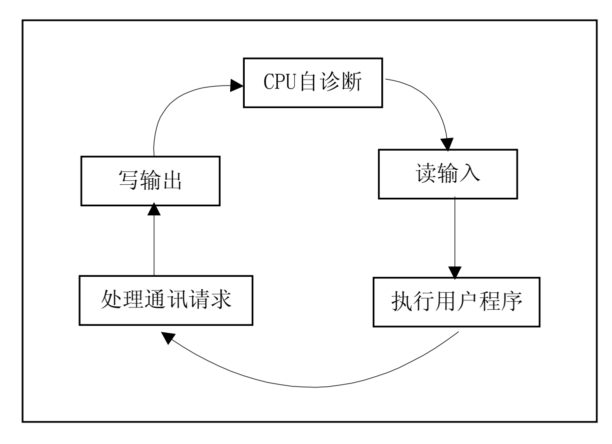


图 3-2 扫描周期

中断事件可能发生在扫描过程的任一时刻，这时候 CPU 将会先暂时中断主循环，转去执行中断服务程序，中断服务程序执行完成后，再在主循环的断点重新进入主循环。如图 3-3。

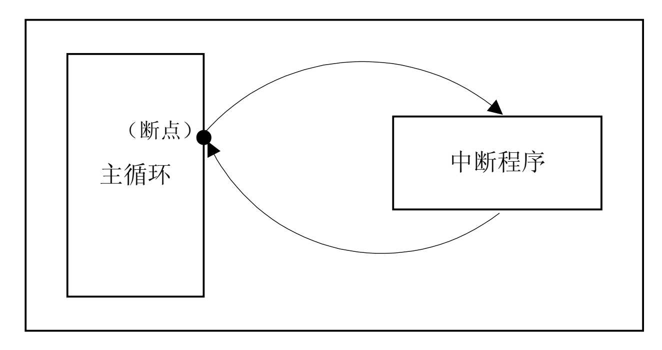


图 3-3 中断服务程序的执行

第四章 EasyProg 软件的使用

本章对 EasyProg 软件界面的组成以及各部分的功能、使用进行了详细的描述，用户在掌握上一章介绍的基本概念的基础上，通过阅读本章可以快速认识并理解 EasyProg 的具体功能以及操作。

LD 编辑器和 IL 编辑器的使用将涉及到 IEC61131-3 标准中的许多语法，在本章中未作介绍，相关内容以及语法将在后续章节中进行详细描述。

4.1 界面总体介绍

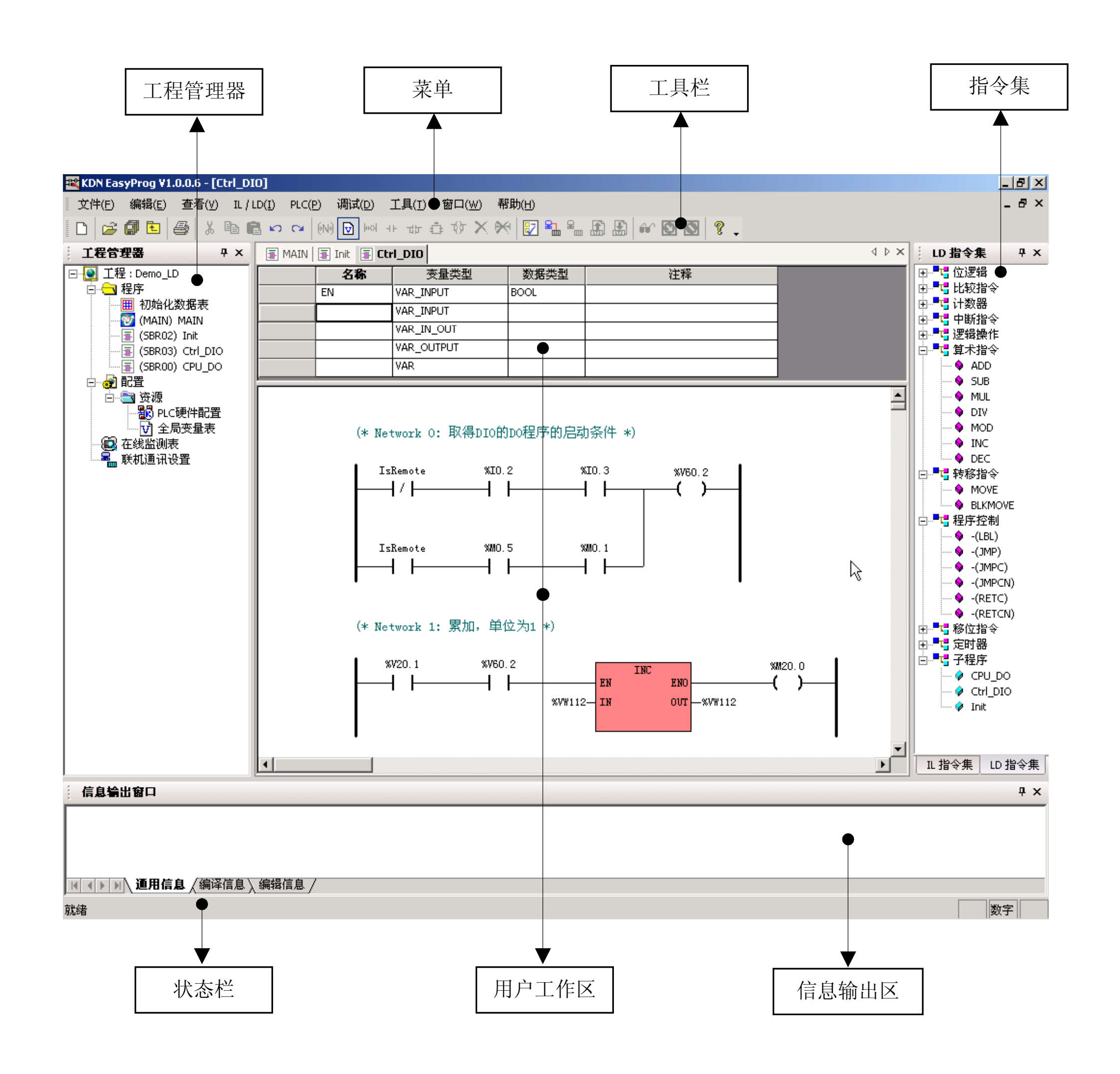


图 4-1 界面的总体组成

- 菜　　单：菜单中包含了 EasyProg 软件所有的操作命令。
- 工　具　栏：工具栏中包含了用户使用频度较高的一些操作命令。
- 状　态　栏：状态条提供了软件当前的状态信息和操作命令的提示信息。

- 工程管理器：工程管理器中采用树型结构显示了整个工程的组织结构，包括程序、配置等。用户可以在此对当前工程进行操作、管理。工程管理器中支持右键菜单。
- 用户工作区：这是用户使用的主要区域，用户可以在此打开硬件配置窗口、全局变量表、编辑器等窗口，完成配置硬件、声明全局变量、编辑程序等任务。

  图 4-1 显示的是编辑器，包括区域上部的变量定义表格和区域下部的程序编辑器。编辑器又分为 LD 编辑器、IL 编辑器。
- 指　令　集：以树状列出了 K3 系列 PLC 支持的所有指令、功能、功能块和用户自己编写的子程序。指令集又分为 LD 指令集、IL 指令集。
- 信息输出区：用于显示软件输出的提示信息，包括编译信息、查找结果等。

4.2　菜单命令介绍

4.2.1 【文件】菜单



- 〔新建工程…〕

建立一个新工程。

用户可通过鼠标单击此菜单或者工具栏上的图标或者快捷键 Ctrl+N 来执行。

执行该命令后，将弹出对话框如下图。选择好路径并输入工程名后，单击〔保存〕按钮即可。

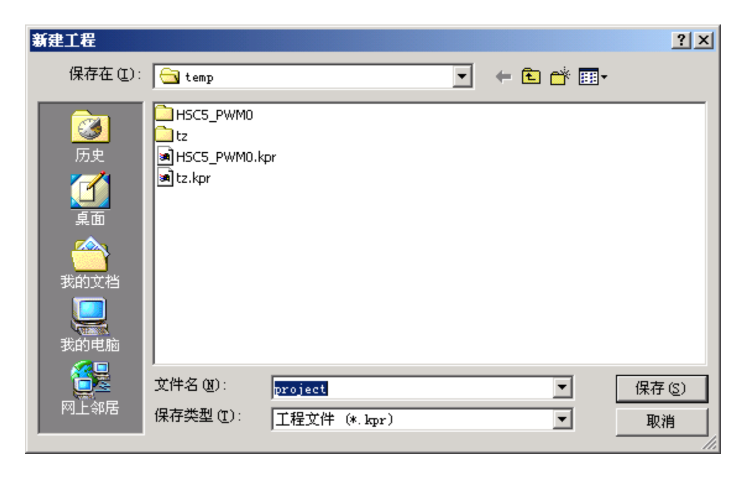


图 4-2　新建工程

- 〔打开工程…〕

打开一个已经存在的工程。

用户可通过鼠标单击此菜单或者工具栏上的图标或者快捷键 Ctrl+O 来执行。

执行该命令后，将弹出对话框如下图。选择好要打开的工程后，单击〔打开〕按钮即可。

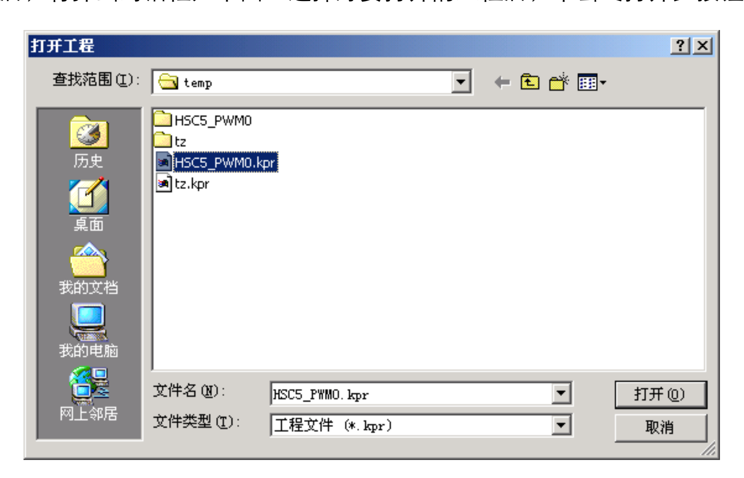


图 4-3　打开工程

• 〔保存工程〕

保存当前正打开的工程，工程名称不变。

用户可通过鼠标单击此菜单或者工具栏上的图标或快捷键 Ctrl+S 来执行。

• 〔工程另存为…〕

将当前已经打开的工程重新命名并以新名字保存。

用户可通过鼠标单击此菜单来执行。执行该命令后，将弹出对话框如下图。

选择好路径并输入工程文件名后，单击〔保存〕按钮即可。

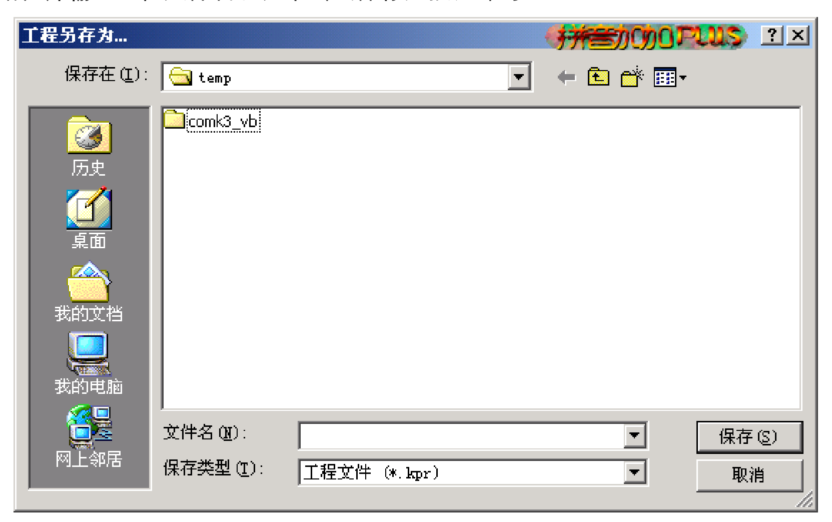


图 4-4 工程另存为…

• 〔关闭工程〕

关闭当前打开的工程；

用户可通过鼠标单击此菜单或者工具栏上的图标或快捷键 Ctrl+F4 来执行。

• 〔导入工程…〕

导入一个已存在的工程备份文件（扩展名为.zip）并将其打开。

用户可通过鼠标单击此菜单来执行命令。

① 执行该命令后，首先将弹出“导入工程…”对话框，如下图。

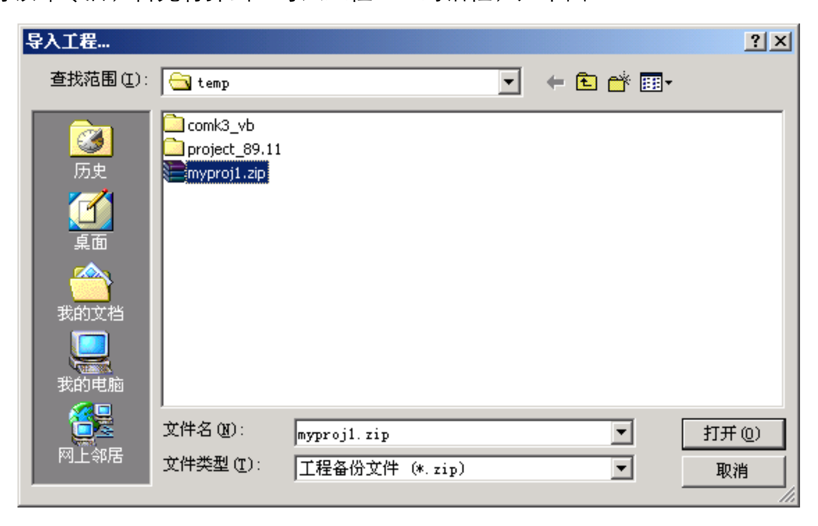


图 4-5

② 选择好备份文件后并单击〔打开〕按钮，将弹出对话框，以选择备份文件解压之后得到的工程文件的存放目录，如下图。

选择好路径后，单击〔确定〕按钮即可将工程文件解压到选定的目录下并将其打开。



图 4-6



若在选定的目录下存在与压缩文件中所包含的工程同名的工程，则会被直接覆盖。

- 〔导出工程…〕

将当前打开的工程所涉及到的所有文件压缩至一个文件（扩展名.zip）中以便于用户备份。

用户可通过鼠标单击此菜单来执行命令。

执行后弹出“导出工程…”对话框，如下图所示。

选择好路径并输入备份文件名后，单击〔保存〕按钮即可。

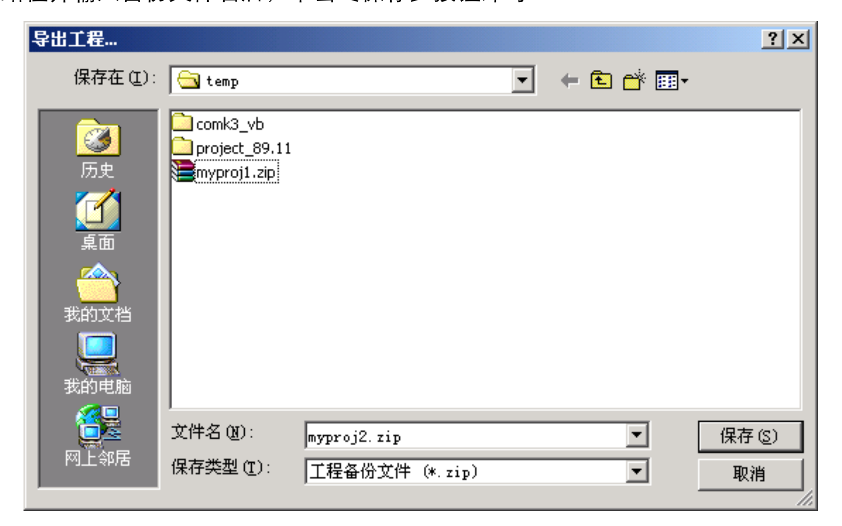


图 4-7　导出工程

在压缩文件中，所有的文件名、相对路径以及工程的相关设置均保持不变。例如，工程名称为 project，工程文件为 project.kpr，则压缩文件中的工程文件仍旧为 project.kpr，工程名称仍旧为 project。

- 〔新建程序…〕

在当前的工程中新建一个程序（主程序、子程序或者中断服务程序，IL 或者 LD 语言）。

用户可通过鼠标单击此菜单或者工具栏上的图标来执行。

① 首先弹出〔新建程序…〕对话框如图 4-8。

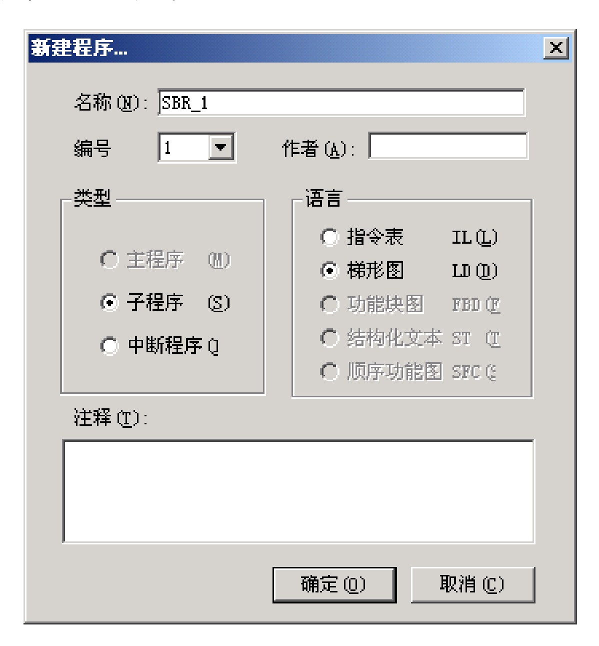


图 4-8　“新建程序…”对话框

- ►〔名称〕：输入该程序的名称。取名的原则请参见 3.4.1 标识符的定义。
- ►〔编号〕：为该程序任选一个编号。
- ►〔作者〕：输入作者的名字。可选。
- ►〔类型〕：选择该程序的类型：主程序、子程序、中断服务程序。

  若工程中没有主程序，将只允许选择“主程序”。

  若工程中已存在主程序，则不允许选择“主程序”。
- ►〔语言〕：选择该程序所用的语言：指令表（IL）、梯形图（LD）。

② 对上述各项作好选择后，单击〔确定〕按钮即可进入 IL 或者 LD 编辑器开始编程。

- 〔最近文件列表〕

在此处列出了用户最近使用过的 4 个工程文件。

用户可以直接单击列表中的文件名称来打开相应的工程。

- 〔退出〕

退出本软件。

用户可通过鼠标单击此菜单或者快捷键 Alt+F4 来执行。

4.2.2 【编辑】菜单

【编辑】菜单中的各命令主要在编写 IL、LD 程序时使用。



- 〔撤销〕

撤销最近一次的操作。

用户可通过鼠标单击此菜单或者快捷键 Ctrl+Z 来执行。

- 〔重复〕

重复最近一次撤销的操作。

用户可通过鼠标单击此菜单或者快捷键 Ctrl+Y 来执行。

- 〔全选〕

用于选定编辑区中所有可编辑的内容。

用户可通过鼠标单击此菜单或者快捷键 Ctrl+A 来执行。

- 〔剪切〕

删除选中的内容，并将选中的内容复制至剪贴板中。

用户可通过鼠标单击此菜单或者快捷键 Ctrl+X 来执行。

- 〔复制〕

将选中的内容复制至剪贴板中。

用户可通过鼠标单击此菜单或者快捷键 Ctrl+C 来执行。

- 〔粘贴〕

将剪贴板中剪切或者复制的内容放至编辑器中。

用户可通过鼠标单击此菜单或者快捷键 Ctrl+V 来执行。

- 〔查找〕

用于在当前活动的 IL 文档中查找指定的字符串。

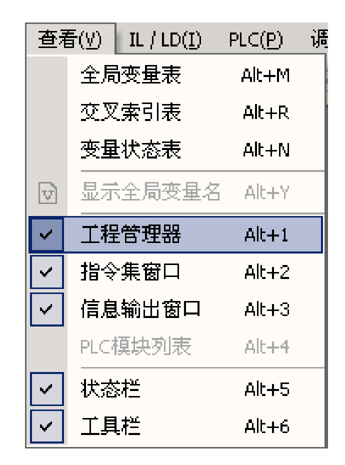
用户可通过鼠标单击此菜单或者快捷键 Ctrl+F 来执行。执行后将弹出“查找…”对话框。

- 〔替换〕

用于在当前活动的 IL 文档中查找指定的字符串，并用指定的字符串来代替查找到的结果。

用户可通过鼠标单击此菜单或者快捷键 Ctrl+R 来执行。执行后将弹出“替换…”对话框。

4.2.3 【查看】菜单



- [全局变量表]

  打开全局变量表窗口。关于全局变量表的详细说明请参见 4.9 全局变量表 部分。

  用户可通过鼠标单击此菜单或者快捷键 Alt+M 来执行。

- [交叉索引表]

  打开交叉索引表窗口。关于交叉索引表的详细说明请参见 4.10 交叉索引表 部分。

  用户可通过鼠标单击此菜单或者快捷键 Alt+R 来执行。

- 〔变量状态表〕

  打开变量状态表视图。关于变量状态表的详细说明请参见 4.11 变量状态表 部分。

  用户可通过鼠标单击此菜单或者快捷键 Alt+N 来执行。

- 〔显示全局变量名〕

  复选。在 IL、LD 编辑器中切换显示全局变量名或者直接 I/O 地址。

  用户可通过鼠标单击此菜单或者快捷键 Alt+Y 来执行。

- 〔工程管理器〕

  复选。切换是否显示工程管理器窗口。

  用户可通过鼠标单击此菜单或者快捷键 Alt+1 来执行。

- 〔指令集窗口〕

  复选。切换是否显示指令集窗口。

  用户可通过鼠标单击此菜单或者快捷键 Alt+2 来执行。

- 〔信息输出窗口〕

  复选。切换是否显示信息输出窗口。

  用户可通过鼠标单击此菜单或者快捷键 Alt+3 来执行。

- 〔PLC 模块列表〕

  复选。切换是否显示 PLC 模块列表。在进行硬件配置时该命令方能生效；

  用户可通过鼠标单击此菜单或者快捷键 Alt+4 来执行。

- 〔状态栏〕

  复选。切换是否显示状态栏。

  用户可通过鼠标单击此菜单或者快捷键 Alt+5 来执行。

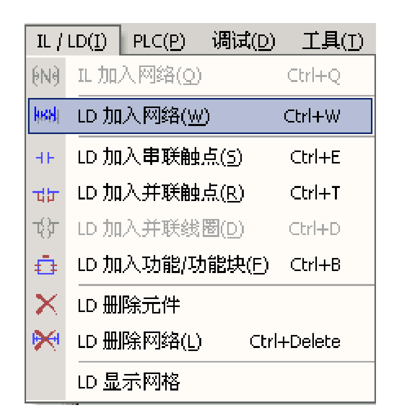
- 〔工具栏〕

  复选。切换是否显示工具栏。

  用户可通过鼠标单击此菜单或者快捷键 Alt+6 来执行。

4.2.4 【IL/LD】菜单

【IL/LD】菜单中的所有菜单命令仅当编辑 IL、LD 程序时方能生效。



- 〔IL 加入网络〕

  在 IL 程序中加入一个网络（Network）。

  用户可通过鼠标单击此菜单或者快捷键 Ctrl+Q 来执行。

- 〔LD 加入网络〕

  在 LD 程序中在选中的元件所在的网络之后加入一个网络（Network）。

  用户可通过鼠标单击此菜单或者快捷键 Ctrl+W 来执行。

- 〔LD 加入串联触点〕

  在 LD 程序中以选中的元件为基准加入一个串联触点。

  用户可通过鼠标单击此菜单或者快捷键 Ctrl+E 来执行。

- 〔LD 加入并联触点〕

  在 LD 程序中以选中的元件为基准加入一个并联触点。

  用户可通过鼠标单击此菜单或者快捷键 Ctrl+T 来执行。

- 〔LD 加入并联线圈〕

  在 LD 程序中以选中的线圈为基准加入一个并联线圈。

用户可通过鼠标单击此菜单或者快捷键 Ctrl+D 来执行。

- 〔LD 加入功能/功能块〕

在 LD 程序中以选中的元件为基准加入一个串联的功能或者功能块（包含用户编写的子程序）。

用户可通过鼠标单击此菜单或者快捷键 Ctrl+B 来执行。

- 〔LD 删除元件〕

在 LD 程序中删除一个选中的触点、线圈、功能或者功能块。

用户可通过鼠标单击此菜单或者快捷键 Delete 来执行。

- 〔LD 删除网络〕

在 LD 程序中删除一个选中元件所在的梯级。

用户可通过鼠标单击此菜单或者快捷键 Ctrl+Delete 来执行。

- 〔LD 显示网格〕

复选。切换是否在 LD 编辑器中显示网格。

4.2.5 【PLC】菜单



- 〔编译工程〕

编译当前工程，生成 PLC 可执行的代码。

用户可通过鼠标单击此菜单或者快捷键 F7 来执行。

- 〔联机〕

将当前编程设备与 PLC 进行连接。

用户可通过鼠标单击此菜单或者快捷键 F2 来执行。

- 〔取消联机〕

取消编程设备与 PLC 当前的联机状态。

该命令在联机成功后方能生效。用户可通过鼠标单击此菜单或者快捷键 Ctrl+F2 来执行。

- 〔上载…〕

将 PLC 中应用工程数据，包括程序、配置等，上载到当前工程中并打开。

该命令在联机成功后方能生效。用户可通过鼠标单击此菜单或者快捷键 Ctrl+F8 来执行。

执行此命令后，将首先弹出对话框如下图所示。

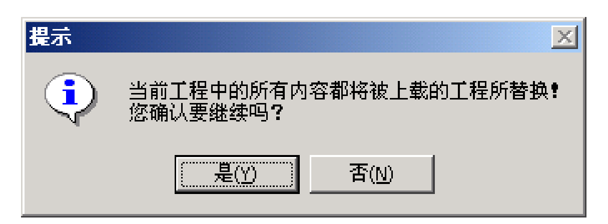


图 4-9

►单击〔是(Y)〕，将继续上载工程并以上载的内容完全替换当前打开的工程中的内容。

►单击〔否(N)〕，则将停止上载。

- 〔下载…〕

将当前工程下载至 PLC 中。

该命令在联机成功后方能生效。用户可通过鼠标单击此菜单或者快捷键 F8 来执行。

为保证下载到 CPU 中的程序是最新的，下载之前会自动进行编译。

在下载时 CPU 必须处于停止状态，因此下载之前会自动查询 CPU 的当前状态并进行提示。

- 〔比较…〕

比较当前打开的工程与 CPU 内正在运行的工程是否一致。

该命令在联机成功后方能生效。用户可通过鼠标单击此菜单来执行。

- 〔清除…〕

将 CPU 中的工程信息（包括程序、配置等）完全删除，并将所有的内存区域复位。

该命令在联机成功后方能生效。用户可通过鼠标单击此菜单来执行。

- 〔修改密码…〕

修改当前 CPU 的保护密码以及保护等级。执行后的窗口如下图所示。

该命令在联机成功后方能生效。用户可通过鼠标单击此菜单来执行。

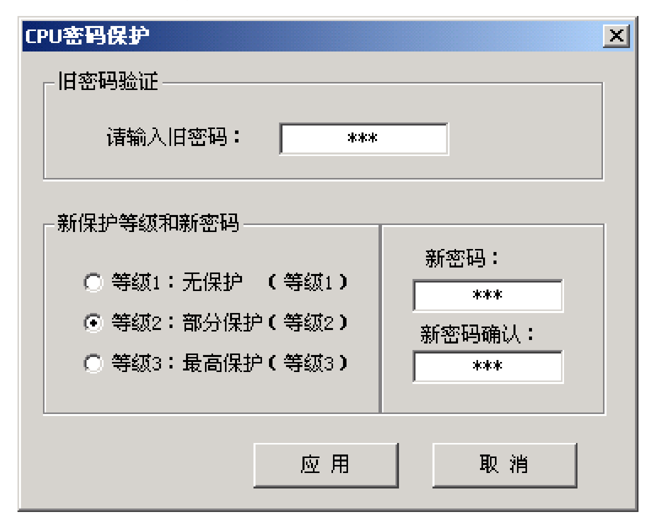


图 4-10　修改保护密码

▶〔旧密码验证〕：输入当前 CPU 的旧密码。在修改密码之前，CPU 将首先验证输入的旧密码与现有的保护密码是否一致，若一致才允许修改密码。

▶ 保护等级：选择 CPU 的密码保护等级，共分三级。

▶〔新密码〕：输入将要设置的新密码。密码的长度最大允许 8 个字符（不支持中文密码）。

▶〔新密码确认〕：再次输入将要设置的新密码进行确认。

▶〔应用〕：单击此按钮将把新密码和新保护等级写入 CPU 中。

▶〔取消〕：单击此按钮将取消当前的输入并关闭本窗口。

• 〔模块信息…〕

读取并显示所连 CPU 的相关信息，包括版本、硬件错误信息等。执行后的窗口如图 4-11 所示。该命令在联机成功后方能生效。用户可通过鼠标单击此菜单来执行。

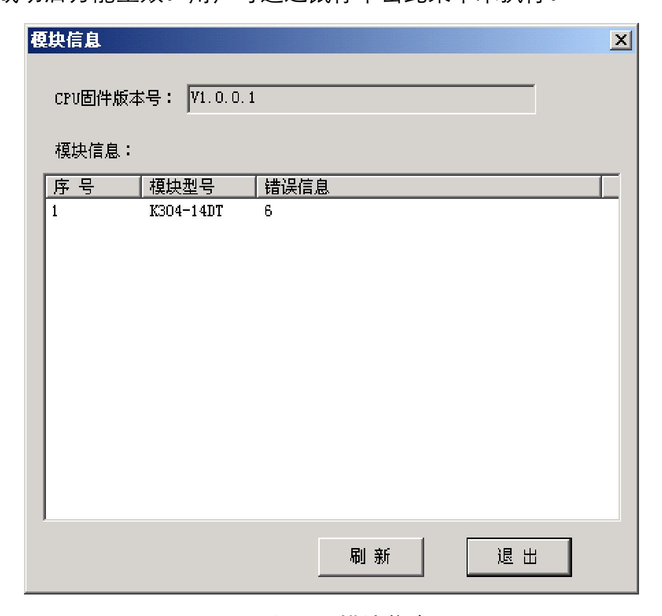


图 4-11 模块信息

▶〔CPU 固件版本号〕：显示当前 CPU 固件（firmware）的版本号。

▶〔模块信息〕：显示当前 CPU 及其所连所有模块的型号和诊断到的硬件错误信息。

▶〔刷新〕：单击此按钮，将立即从 CPU 中读取一次模块信息并对窗口的显示进行刷新。

▶〔退出〕：单击此按钮，将退出此窗口。

- 〔调整 CPU 时钟…〕

读取并可修改 CPU 内的实时时钟值。执行后的窗口如下图所示。

该命令在联机成功后方能生效。用户可通过鼠标单击此菜单来执行。



图 4-12 调整 CPU 时钟

▶〔系统当前时间〕：显示编程所用 PC 机当前的日期、时间。

▶〔PLC 当前时间〕：显示所连 CPU 内实时时钟的当前时间。

若背景色是绿色，代表成功地从 CPU 内读到了实时时钟。

若背景色是黄色，代表从 CPU 内读取实时时钟失败。

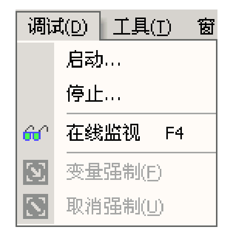
▶〔将 PLC 时间调整为〕：用户可以在此输入将要设置的 CPU 实时时钟的新时间值。

▶〔修改时钟〕：单击此按钮，用户输入的新的时间值将被写入 CPU 内的实时时钟中。

▶〔读取时钟〕：单击此按钮，将从 CPU 内读取实时时钟值并刷新〔PLC 当前时间〕显示。

▶〔关闭〕：单击此按钮，将取消所有的输入并退出本窗口。

4.2.6 【调试】菜单



- 〔启动…〕

远程启动 CPU。

该命令在联机成功后方能生效。用户可通过鼠标单击此菜单来执行。

- 〔停止…〕

远程停止 CPU。

该命令在联机成功后方能生效。用户可通过鼠标单击此菜单来执行。

- 〔在线监测〕

复选，切换进入或者退出在线监测状态。

在线监测时，EasyProg 软件将连续从 CPU 内读取当前工程中用到的所有变量并显示。

若 EasyProg 中当前打开的工程与 CPU 内运行的工程不一致，则不允许进入在线监测状态。

该命令在联机成功后且当前视图是 IL、LD 编辑器或者变量状态表时方可生效。

用户可通过鼠标单击此菜单或者快捷键 F4 来执行。

- 〔变量强制〕

将变量状态表中所输入的“地址”置为强制状态，并将其值强行设置为输入的“强制值”。

若“地址”对应的“强制值”没有输入，则不对该“地址”进行强制。

该命令在联机成功后且当前视图为变量状态表时方可生效。用户可通过鼠标单击此菜单来执行。



注意：KDN-K3 系列 CPU 采用强制优先模式。

比如，若 I1.0 处于强制状态，则其强制值将生效，而来自现场的输入将无效!

- 〔取消强制〕

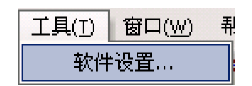
取消 CPU 内所有变量的强制状态。

该命令在联机成功后方可生效。用户可通过鼠标单击此菜单来执行。



由于 KDN-K3 系列 CPU 采用强制优先模式，一定要记得在调试结束后取消强制状态!

4.2.7 【工具】菜单



- 〔软件设置…〕

对 EasyProg 软件进行配置。

用户可通过鼠标单击此菜单来执行。执行此命令后，将弹出“软件设置”对话框。对话框内分为如下几个页面：

① 〔界面风格〕页面

此处允许对界面的外观风格进行设置。

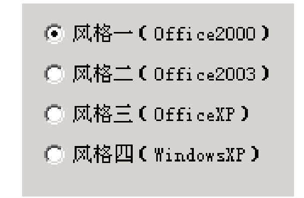


图 4-13 〔界面风格〕页面

界面风格有如下四种选择：

- 风格一（Office2000）：仿 Microsoft Office 2000 的界面样式。
- 风格二（Office2003）：仿 Microsoft Office 2003 的界面样式。
- 风格三（OfficeXP） ：仿 Microsoft Office XP 的界面样式。
- 风格四（Windows XP）：在 Windows XP 操作系统下建议用户选择此风格。

② 〔编辑器〕页面

此处允许对编辑器的某些环境变量，包括字体、颜色、数据显示格式等，进行配置。



图 4-14 〔编辑器〕页面

- 〔在线监测时数据显示格式〕

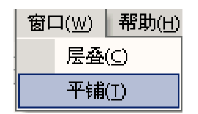
  选择当在线监测时整数的显示格式。有如下三种选项：

  〔混合〕：INT、DINT 型数据以十进制格式，BYTE、WORD、DWORD 型数据以十六进制格式显示。

  〔十 进 制〕：所有整数都以十进制格式显示。

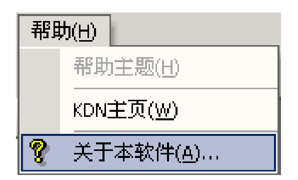
  〔十六进制〕：所有整型数据都以十六进制格式显示。

4.2.7 【窗口】菜单



这些是标准的 Windows 命令，不再赘述。

4.2.8 【帮助】菜单



这些命令非常简单，此处不再赘述。

4.3 工具栏介绍



在工具栏中集成了经常使用的功能选项，每一个工具按钮都对应着一个菜单命令，为用户的开发提供了非常大的便利。

当把光标移到某个工具按钮图标上并稍微停留一会，就会弹出信息框来提示该按钮的功能，同时在状态栏上也会有提示。

- ：新建工程或者程序。

若在软件中当前没有打开的工程，单击此按钮将新建工程；若当前有打开的工程，单击此按钮将新建一个程序。

4.4 工程管理器介绍

工程管理器是界面中的主要窗口之一，以树状列表的形式直观地显示出了当前工程的所有组成部分，包括程序、硬件配置、变量状态表、全局变量表等。用户可以在此窗口中对当前打开的工程进行管理、操作、维护。

工程管理器的各个树节点均支持右键，用右键单击某个节点将会弹出相应的右键菜单来。

工程管理器的外观如图 4-15 所示。

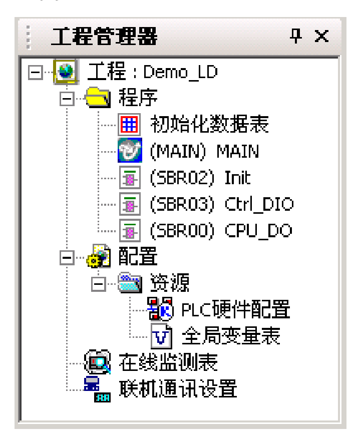


图 4-15 工程管理器

4.4.1 浮动的工程管理器窗口

工程管理器采用了浮动窗口技术，用户可以根据需要方便地进行切换使其浮动或者在固定位置显示，方法如下：

① 方法一

执行【查看】→〔工程管理器〕菜单命令，即可进入浮动状态并切换使其隐藏或者显示。

② 方法二

• 单击窗口右上角的 图标，即可让窗口进入浮动状态，并缩成一个图标停靠在屏幕左边，图标的标题是〔工程管理器〕。

• 在浮动状态下，将鼠标置于屏幕左边的〔工程管理器〕图标上并停留片刻，工程管理器窗口就会出现在屏幕上并允许用户进行正常的操作，然后若将鼠标置于工程管理器窗口之外，它又将收缩于屏幕左侧。

• 在浮动状态下，单击工程管理器窗口右上角的图标，就会重新回复至固定位置显示。

4.4.2 【程序】组

4.4.2.1 描述

在【程序】组下列出了当前工程中包含的所有程序块，包括：

① 初始化数据表（可选）

用户可在初始化数据表中为 V 区中 BYTE、WORD、DWORD、INT、DINT、REAL 类型的数据指定初始值。初始化数据表是工程中的可选部分，用户也可以不在表中输入任何内容。

关于初始化数据表的更详细的描述请参见下面 4.6 初始化数据表 部分。

② 主程序

主程序是在 CPU 内运行的主循环任务，在 CPU 的每个扫描周期内都会被执行一次。

请参阅前文 3.8 CPU 中程序的执行 部分。

在工程中有且仅有 1 个主程序。主程序必须在其它程序之前创建，且创建之后不能删除。

③ 中断服务程序（可选）

中断任务，当指定的中断事件发生时方会被执行。

请参阅前文 3.8 CPU 中程序的执行 部分。

在一个工程中最多可有 32 个中断服务程序。中断服务程序可以从工程中删除，其名称也可以修改。

④ 子程序（可选）

子程序是可重用的代码段，是为了方便用户进行结构化程序设计而提供的，用户可以将

常用的控制功能编写成一个个子程序。

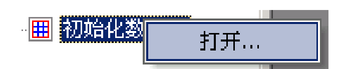
子程序仅供主程序和中断服务程序调用，只有被调用的子程序才能在 CPU 中得以运行。

在一个工程中子程序最多可有 32 个。子程序可以从工程中删除，其名称也可以修改。

4.4.2.2 操作方法

① 初始化数据表

在〔初始化数据表〕节点上单击鼠标右键，将会弹出如下右键菜单：



执行此〔打开…〕菜单命令，或者双击〔初始化数据表〕节点，都将打开初始化数据表。

② 【程序】组

在【程序】组节点上单击鼠标右键，将会弹出如下右键菜单：



执行此〔新建…〕菜单命令，将开始新建一个程序。其作用与【文件】→〔新建程序…〕菜单命令的作用是一样的。

③ 各个程序节点

在各个程序的名称上单击鼠标右键，将会弹出如下右键菜单：



- 〔打开〕：打开该程序。双击该程序名称，也可以打开该程序。
- 〔删除〕：从工程中删除该程序。
- 〔重命名〕：为该程序重新命名。
- 〔属性〕：打开“程序属性“对话框，如图 4-16，可以对该程序的属性进行修改。

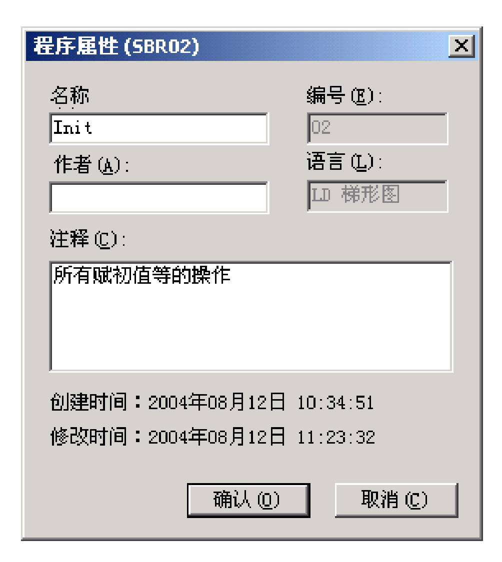


图 4-16　“程序属性”对话框

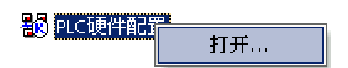
4.4.3 【配置】→【资源】组

所谓资源，主要是指应用系统的硬件构成。

【资源】组中包含了〔PLC 硬件配置〕和〔全局变量表〕两个子项。

4.4.3.1 〔PLC 硬件配置〕

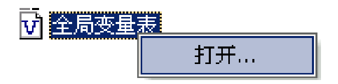
在〔PLC 硬件配置〕节点上单击鼠标右键，将会弹出如下右键菜单：



执行此〔打开…〕命令，或者双击〔PLC 硬件配置〕节点，均可以进入硬件配置窗口。

4.4.3.2〔全局变量表〕

在〔全局变量表〕节点上单击鼠标右键，将会弹出如下右键菜单：



执行此〔打开…〕命令，或者双击〔全局变量表〕节点，均可以进入全局变量定义窗口。

4.4.4 【变量状态表】

在〔变量状态表〕节点上单击鼠标右键，将会弹出如下右键菜单：

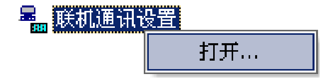


执行此〔打开…〕命令，或者双击〔变量状态表〕节点，均可以进入变量状态表窗口。

4.4.5 【联机通讯设置】

4.4.5.1 描述

在〔联机通讯设置〕节点上单击鼠标右键，将会弹出如下右键菜单：



执行此〔打开…〕命令，或者〔联机通讯设置〕节点，均可进入“编程设备联机通讯设置”窗口，对当前编程设备的串行通讯口的通讯参数进行设置，以便与 PLC 联机。

4.4.5.2 “编程设备联机通讯设置”窗口描述

编程设备通讯配置
本机地址： 0
PLC站号： 1
确 定
取 消
COM口： COM1
波特率： 19200
奇偶校验： 无校验
数据位： 8
停止位： 1

图 4-17 编程设备联机通讯设置

- 〔本机站号〕：当与 K3 系列 PLC 联机时，本机作为主站，其站号固定为 0。
- 〔PLC 站号〕：选择将要连接的 CPU 的站号，该站号即 CPU 作为 Modbus 从站时的站号。
- 〔COM 口〕 ：选择本机将要使用的 COM 口。
- 〔波特率〕 ：设置串行通讯的波特率，其值有：2400、4800、9600、19200、38400。
- 〔奇偶校验〕：设置串行通讯的奇偶校验方式，其值有：无校验、奇校验、偶校验。
- 〔数据位〕 ：设置串行通讯的数据位，其值有：7、8、9。
- 〔停止位〕 ：设置串行通讯的停止位，其值有：1、2。

若没有进入该窗口进行设置，则 EasyProg 软件默认使用的联机通讯参数为：将要连接的 CPU 的〔PLC 站号〕为 1；本机使用 COM1 口，波特率 19200，无校验，8 位数据位，1 位停止位。

4.5 指令集窗口

在指令集窗口中以树状列表的形式列出了 KDN-K3 系列 PLC 支持的所有指令，目的是为用户

的编程提供方便。指令集又分为 IL 指令集和 LD 指令集两大类。目前“IL 指令集”仅供用户浏览，没有实际操作。在进行 LD 编程时可以操作 LD 指令集。指令集窗口的外观如图 4-18。

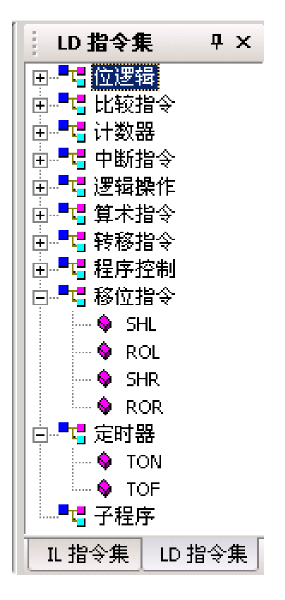


图 4-18　指令集窗口

4.5.1 浮动的指令集窗口

指令集窗口支持浮动技术，具体的操作方法请参见 4.4.1 浮动的工程管理器窗口 部分。

4.5.2 在编程时使用指令集窗口

当编辑 LD 程序时，可按如下步骤来使用指令树：

- 单击当前程序中的某个元件，该元件将被选中作为基准；
- 展开 LD 指令树中相应的文件夹，双击要加入的指令即可将该指令加入到程序中。

4.6　信息输出窗口

信息输出窗口用于显示 EasyProg 软件提示的各种信息，其中“编译信息”窗口显示了用户

最近一次的编译信息，而“通用信息”窗口显示了其它一些提示信息，比如查找结果等。

信息输出窗口的外观如图 4-19。

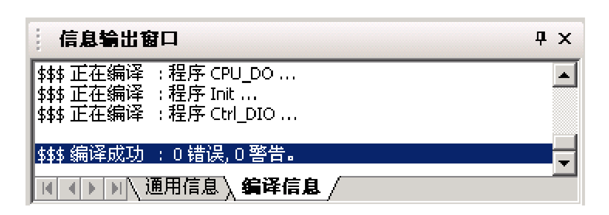


图 4-19　信息输出窗口

4.6.1 浮动的信息输出窗口

信息输出窗口支持浮动技术，具体的操作方法请参见 4.4.1 浮动的工程管理器窗口 部分。

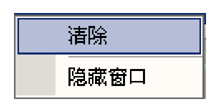
4.6.2　在信息输出窗口中进行操作

① 错误定位

若当前工程编译有错误，则双击“编译信息”窗口中的错误信息就会自动打开相应的程序并定位到错误所在的位置，其中：IL 程序将定位到行，LD 程序将定位到网络（Network）。

② 右键菜单

在输出区内任意位置单击鼠标右键，将弹出如下右键菜单：



- 〔清除〕　：清除当前输出区内的所有内容。
- 〔隐藏窗口〕：隐藏（或者称浮动）信息输出窗口。

4.7 PLC 硬件配置

EasyProg 软件为用户提供了功能完善、使用灵活方便的硬件配置环境，用户可以根据需要

在此对工程中所用的 PLC 进行配置，包括定义用到的 PLC 模块，配置各个模块的具体参数等。硬件配置窗口的样式如下图。从图中可以看出，窗口可分为两部分：

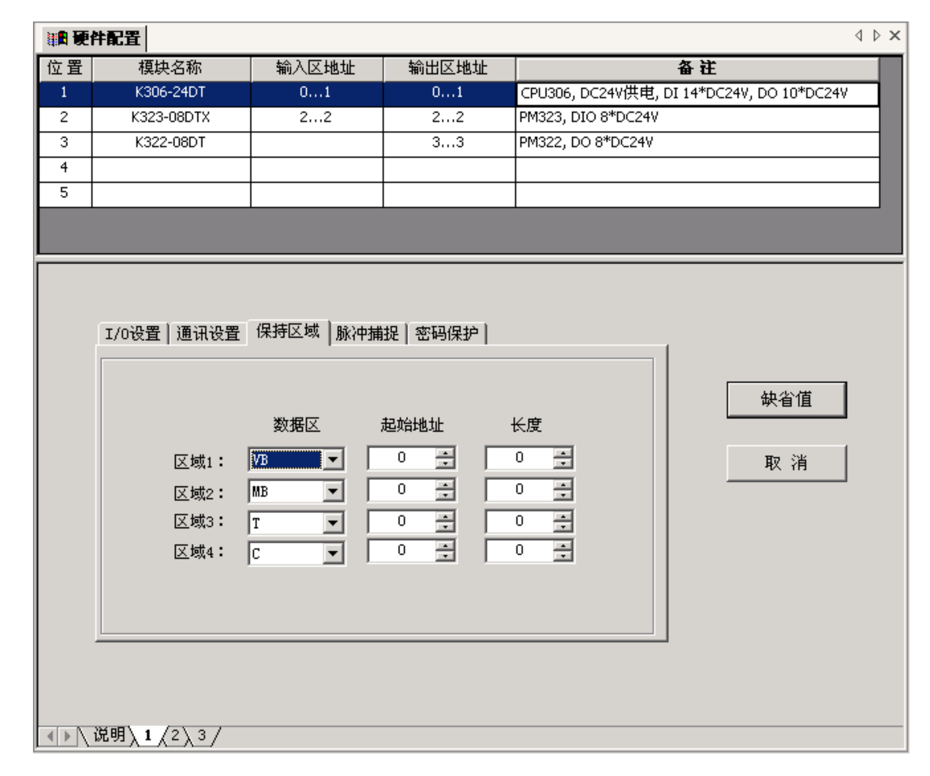


图 4-20 硬件配置窗口

- PLC 模块列表

以表格形式列出了本工程中用到的 PLC 模块，并允许用户进行配置。PLC 模块在表格中的排列次序应当与实际模块在背板总线上的连接次序一致。

- PLC 模块参数配置窗口

在该窗口可以对列表中各 PLC 模块的实际参数进行配置。

硬件配置信息必须下载到 CPU 中方可生效。在每次冷启动时，CPU 会首先检测实际所连模块的信息，并将检测到的实际信息与储存的配置信息进行比较，若有错误就会转入 STOP 状态，

否则就进入正常运行状态。

硬件配置是工程中必需的信息，另外，根据所配 CPU 的不同，系统允许访问的 I/O 区域也有所不同，因此建议用户在工程中首先完成硬件配置。当用户新建一个工程时，EasyProg 将缺省地加入一个 K306-24DT 型号的 CPU。

4.7.1 如何进入硬件配置窗口?

有如下两种方式可以进入硬件配置窗口:

- 双击工程管理器中【资源】组下的〔PLC 硬件配置〕节点;
- 在〔PLC 硬件配置〕节点上单击鼠标右键，然后执行弹出的〔打开…〕菜单命令。

4.7.2 添加、删除模块

- 添加模块

鼠标单击 PLC 模块列表的表格中将要加入模块的位置，将焦点置于该行，然后在右侧的〔PLC 模块列表〕中双击要加入的模块即可。若该行已有模块存在，则必须首先删除已存在的模块，然后才能够加入新的模块。

第 1 行（“位置”为 1 的行）只能加入 CPU 模块，其余各行只能加入扩展 I/O、功能模块。各个模块之间不允许有空行存在。若有空行，则软件不允许在其后添加模块，并且在保存、编译时会提示错误。

CPU304、CPU306、CPU308 所允许的最大 I/O 点数均有明确的规定，若已添加的模块的点数超出了 CPU 所允许的限制，则软件将不允许继续添加模块，并且在保存、编译时会提示错误。

- 删除模块

鼠标单击模块配置表格中将要删除的模块，然后敲 Delete 键删除即可，或者单击鼠标右键，执行〔删除模块〕命令即可。

4.7.3 模块参数的配置

K3 系列 PLC 的每种模块都提供了多种参数和选项设置以适应不同的具体应用，包括模块的 I/O 地址、模拟量模块各通道的信号类型等等，在 EasyProg 软件允许用户自定义所有的这些参数和选项。

鼠标单击模块列表中的某项，或者使用键盘上的向上、向下键移动到某项，均会在下边的 PLC 模块参数配置窗口中自动切换到相应模块的参数配置页面。

在 PLC 模块参数配置区的右侧有两个公用的按钮：〔缺省值〕、〔取消〕。

〔缺省值〕：单击此按钮，将会取消当前页面用户的输入，软件为本模块自动分配参数。

〔取　消〕：单击此按钮，将会取消当前页面用户的输入，恢复本模块原有的配置。

在配置时请注意：各模块在同一区域（I、Q、AI 或者 AQ）内的地址不允许重叠！

4.7.3.1　CPU 参数配置

① 〔I/O 设置〕页面

在此页面中可以完成 CPU 本体 I/O 的相关配置。如下图。



图 4-21　CPU 本体 I/O 参数配置页面

- 输入区

用于配置 CPU 本体集成的 DI 点的参数。

▶〔起始地址〕：DI 部分在 I 区中所占用地址的起始字节，固定为 0。

▶〔输入滤波〕：为部分或者全部的 DI 点选择输入滤波延时，延时范围为〔0.2，12.8〕ms。CPU 上所有的 DI 点以 8 个为单位被分组，系统允许为前两组选择此项配置。来自于现场的 DI 信号至少要保持所配置的延时时间方可被 CPU 认为有效，这样有助于滤除输入噪声，增强系统的抗干扰能力。

- 输出区

用于配置 CPU 本体集成的 DO 点的参数。

▶〔起始地址〕：DO 部分在 Q 区中所占用地址的起始字节，固定为 0。

▶〔输出保持〕：设置当 CPU 处于 STOP 状态时各 DO 点的输出状态。

将某 DO 点对应的复选框设置为☑，则当 CPU 处于 STOP 时，该点输出为 1。

将某 DO 点对应的复选框设置为☐，则当 CPU 处于 STOP 时，该点输出为 0。

此项功能对于 CPU 停机后所需要的安全连锁非常有意义。

② 〔通讯设置〕页面

用于配置 CPU 本体所带串行通讯口（Port0、Port1）的参数。如图 4-22。

I/O设置 | 通讯设置 | 保持区域 | 脉冲捕捉 | 密码保护

Port0 (RS232)
站号：1
波特率：19200
奇偶校验：无校验
数据位：8
停止位：1

Port1 (RS485)
站号：
波特率：
奇偶校验：
数据位：
停止位：

CPU通讯处理时间（5%-50%）：10 %

图 4-22　CPU 本体串口参数配置页面

- Port0

► 〔站号〕　：为 Port0 指定一个唯一的站号。该站号作为 Modbus 从站时的站号。

► 〔波特率〕：设置串行通讯的波特率，其值有：2400、4800、9600、19200、38400。

► 〔奇偶校验〕：设置串行通讯的奇偶校验方式，其值有：无校验、奇校验、偶校验。

► 〔数据位〕：设置串行通讯的数据位，其值有：7、8、9。

► 〔停止位〕：设置串行通讯的停止位，其值有：1、2。

- Port1

请参见上面关于 Port0 的参数的描述。

在 CPU304 和 CPU306 中只有一个串口，不提供 Port1。

- CPU 通讯处理时间

在 3.8 CPU 中程序的执行 部分提到过，在每个扫描周期中 CPU 都要处理用户的通讯请求。〔CPU 通讯处理时间〕就是用来设置通讯处理允许占用 CPU 扫描周期的最大百分比，其范围是〔5%，50%〕。

③ 〔保持区域〕页面

当 CPU 掉电时，在此页面中定义的 4 个内存区域中的数据将由超级电容供电来得到保护，以供再次上电时使用。在常温下保持的时间最长为 72 小时。该页面如下图。

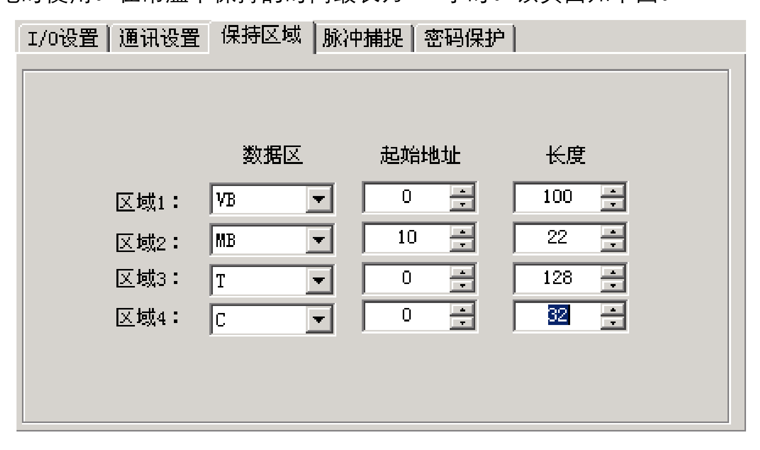


图 4-23 保持区域参数配置页面

- 〔数据区〕 ：指定被保护的数据区所在的内存区，有 V、M、C 四个选项。
- 〔起始地址〕：指定该保持区域的起始字节地址。
- 〔长度〕 ：指定该保持区域的长度，单位：字节。

④ 〔脉冲捕捉〕页面

用于指定 CPU 本体的前 14 个 DI 点是否启用脉冲捕捉功能。

K3 系列 CPU 提供了短脉冲的捕捉功能。由于 CPU 总在每个扫描周期的开始来读取物理 DI 信号的状态并更新输入映像区（I 区），因此，若物理 DI 信号在扫描周期中到达并且只持续非常短的时间时，CPU 有可能丢失这个信号。当一个 DI 点的脉冲捕捉被启用时，物理输入的状态变化会被锁存并一直保持到下一个扫描周期的的 I 区刷新。脉冲捕捉功能提高了 CPU 检测短脉冲信号的可靠性，其作用方式如图 4-24。

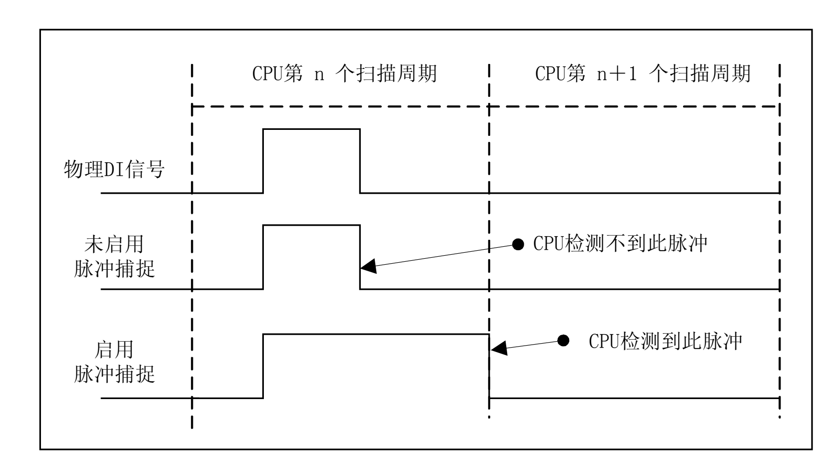


图 4-24　脉冲捕捉功能描述

另外，需要注意的是，物理 DI 信号首先通过输入滤波，然后再脉冲捕捉功能才能生效。因此，当启用脉冲捕捉功能时，必须要保证〔输入滤波〕的时间要小于脉冲的持续时间以免脉冲被滤掉。

〔脉冲捕捉〕页面如下图。



图 4-25　脉冲捕捉参数配置页面

在页面上，若某个 DI 点对应的复选框被选中，则该 DI 点的脉冲捕捉功能被启用，否则就被禁用。如上图，I0.0、I0.1、I0.2、I1.0、I1.1、I1.2 对应通道的脉冲捕捉功能被启用。

4.7.3.2 DI 模块的参数配置

DI 模块的参数配置非常简单，仅需配置其地址即可，如下图所示。

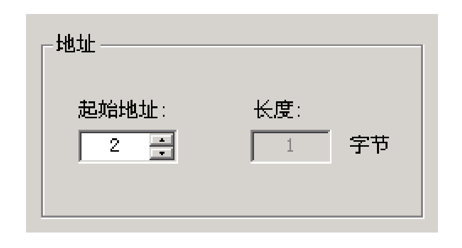


图 4-26 DI 模块参数配置页面

► 〔起始地址〕：指定该 DI 模块所有通道在 I 区中所占用地址的起始字节。

4.7.3.3 DO 模块的参数配置

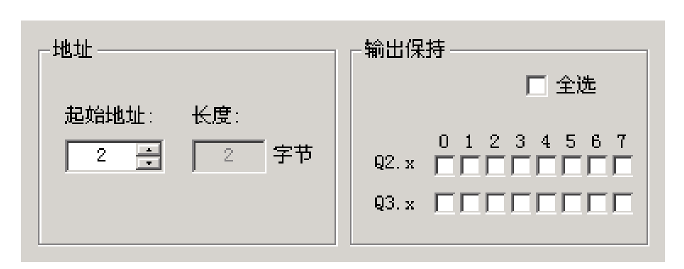


图 4-27 DO 模块参数配置页面

► 〔起始地址〕：指定 DO 模块所有通道在 Q 区中所占用地址的起始字节。

► 〔输出保持〕：设置当 CPU 处于 STOP 状态时该模块各 DO 点的输出状态。

将某 DO 点对应的复选框设置为☑，则当 CPU 处于 STOP 时，该点输出为 1。

将某 DO 点对应的复选框设置为☐，则当 CPU 处于 STOP 时，该点输出为 0。

此项功能对于 CPU 停机后所需要的安全连锁非常有意义。

4.7.3.4　DIO、DI/O 模块的参数配置

DIO 模块是指该模块上的所有通道均可以作为 DI 或者 DO 点使用。模块上的每个点都具有 DI 点、DO 点两种特性，用户不需要作任何配置，直接将其作为 DI 或者 DO 点使用即可。

DI/O 模块是指在该模块上集成了若干个 DI 点和若干个 DO 点，DI、DO 点不能混用。

DIO、DI/O 模块的参数配置界面是一样的，如下图所示。

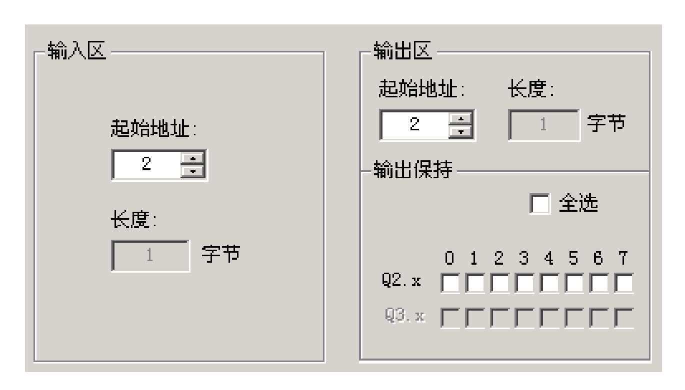


图 4-28　DIO、DI/O 模块参数配置页面

界面上所有参数的含义请参见上面关于 DI、DO 模块配置的说明。

4.7.3.5　AI 模块的参数配置

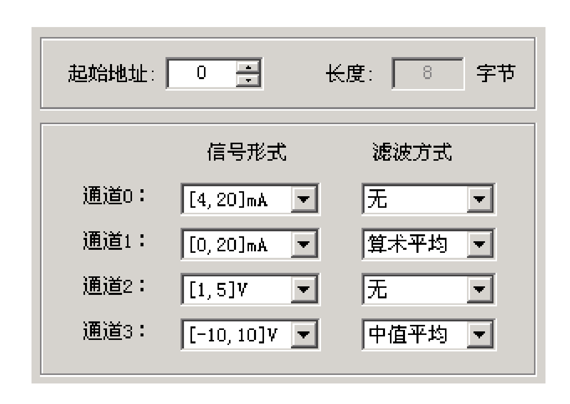


图 4-29　AI 模块参数配置页面

① 参数说明

- ►〔起始地址〕：指定该 AI 模块所有通道在 AI 区中所占用地址的起始字节。
  每个 AI 点在 AI 区占用 2 个字节。因此，地址必须为偶数。
- ►〔信号形式〕：为各个通道选择所接入的信号形式。
  可选的信号形式取决于所选的模块类型，有电流/电压、热电阻、热电偶等。
  测量的信号值将在 CPU 内自动进行线性转换，关于数据转换格式请参见第二部分硬件手册中 6.1.4 信号测量值内部表示格式 相关内容。
- ►〔滤波方式〕：为各个通道选择软件滤波器。
  对于变化较快的模拟量信号使用滤波器可以让其 AI 值变得比较稳定。
  注意：若系统需要对某 AI 信号快速地响应则不应该启用该点的软件滤波器。

② 关于软件滤波器的说明

软件滤波器的输出就是对一定数量的信号采样值取平均值。软件滤波器有如下选项：

- ► *无* --- 不启用软件滤波器。
- ► *算术平均* --- 对一定数量的信号采样值取算术平均值。
- ► *中值平均* --- 将一定数量的信号采样值去掉最大、最小值后，对剩下的数取平均值。

4.7.3.6 AO 模块的参数配置



图 4-30　AO 模块参数配置页面

▶〔起始地址〕：指定该 AO 模块所有通道在 AO 区中所占用地址的起始字节。

每个 AO 点在 AO 区占用 2 个字节。因此，地址必须为偶数。

▶〔信号形式〕：为各个通道选择所接入的信号形式。

可选的信号形式取决于所选的模块类型，有电流/电压等。

关于各种信号数据转换格式的说明请参见 。

▶〔停机保持〕：指定当 CPU 处于 STOP 状态时，该点是否保持原来的输出。

若将该电对应的复选框设置为☑，则该点采用停机保持方式。

若将该电对应的复选框设置为☐，则该点不采用停机保持方式。

▶〔停机输出值〕：若该点不设置为停机保持方式，则在此设置停机时该点的输出值。

此处应当输入经过线性转换之后的数值而非实际的信号值。例如，若用户选择信号形式为〔1,5〕V，停机时希望输出 1V，则由于 PLC 将〔1,5〕V 线性转换为〔1000,5000〕，因此用户应当在此处输入 1000。

4.8 初始化数据表

在初始化数据表中用户可以为 V 区中的 BYTE、WORD、DWORD、INT、DINT、REAL 类型的数据指定初始值。其样式如图 4-31。

初始化数据表

起始地址	数据类型	初始值	初始值	初始值	初始值
%VW0	INT	5	6	7	8
%VB10	BYTE	B#21	B#22		
%VD100	DWORD	DW#2	DW#12		DW#32

图 4-31 初始化数据表

表中共分六列：〔起始地址〕、〔数据类型〕以及连续 4 列〔初始值〕。

一个典型的行的含义是：为从〔起始地址〕开始、指定〔数据类型〕的一个或连续多个（最

多为 4 个）V 区直接变量赋予指定的〔初始值〕。在连续赋值时，〔初始值〕列中不允许为空，否则其后续的〔初始值〕均被认为无效。

如图 4-31 中，第一行的含义是 VW0、VW2、VW4、VW6 均被设置为 INT 型，并分别赋予初值 5、6、7 和 8；第二行的含义是 VB10、VB11 均被设置为 BYTE 型，并分别赋予初值 B#21、B#22；第三行的含义是 VD100、VD104 被设置为 DWORD 型，并分别赋予初值 DW#2、DW#12。

4.8.1 如何进入“初始化数据表”窗口?

有如下两种方式可以进入“初始化数据表”窗口：

- 双击工程管理器中【程序】组下的〔初始化数据表〕节点；
- 在〔初始化数据表〕节点上单击鼠标右键，然后执行弹出的〔打开…〕菜单命令。

4.8.2 在初始化数据表中进行编辑

- 输入数据

鼠标单击某表格单元即可让该单元进入编辑状态。

敲任意键也可以让具有焦点的表格单元进入编辑状态。

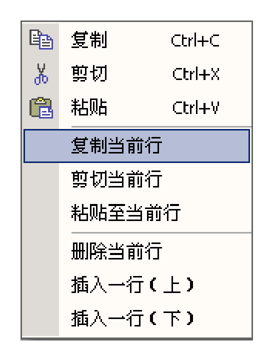
输入结束后，须敲 Enter 键，或者鼠标单击输入单元之外的表格任意地方来进行确认。

若输入的数据有错误，将会自动变成红色进行提示。错误的数据在保存时会被略掉。

用户在输入〔初始值〕时，可以输入任何合法的常量，在需要时软件会根据所选的〔数据类型〕自动对用户输入的数据进行类型转换。

- 右键菜单

在表中任意一个单元单击鼠标右键，将会弹出如下菜单：



- 〔复制〕：将具有焦点的表格单元的内容复制至剪贴板中，快捷键 Ctrl+C。
- 〔剪切〕：删除具有焦点的表格单元的内容，并将其复制至剪贴板中，快捷键 Ctrl+X。
- 〔粘贴〕：将剪贴板中的内容复制至具有焦点的表格单元中，快捷键 Ctrl+V。
- 〔复制当前行〕：复制焦点所在行中各表格单元的内容。
- 〔剪切当前行〕：剪切焦点所在行中各表格单元的内容。
- 〔粘贴当前行〕：将剪贴板中内容复制至焦点所在行的各表格单元中。
- 〔删除当前行〕：删除焦点所在的行。
- 〔插入一行（上)〕：在焦点所在行的上一行的位置插入一个新的空行。
- 〔插入一行（下)〕：在焦点所在行的下一行的位置插入一个新的空行。

4.9 全局变量表

在全局变量表中用户可以方便地完成全局变量的定义，在 IEC61131-3 中，全局变量的关键字是 VAR_GLOBAL。全局变量表窗口分为〔变量〕和〔功能块实例〕两部分页面。

- 〔变量〕页面

用于定义直接存取 PLC 内直接地址的符号变量。样式如下图所示。

VAR_GLOBAL

序号	名称	地址	数据类型	注释
1	Motor1_Err	%I0.0	BOOL	1#电机故障信号
2	Motor1_Run	%I0.1	BOOL	1#电机运行信号
3	Tank1_Level	%AIW0	INT	1#罐液位
4				
5				
6				
7				
8				

变量 / 功能块实例

图 4-32 全局变量表中的〔变量〕页面

符号变量可以等效地代替对应的 PLC 直接地址在程序中使用，从而能够保证用户程序具有良好的可读性。每一个直接地址只允许赋予一个符号变量名，同样地，每一个符号变量名只允许有一个对应的直接地址。

表中分四列：〔名称〕、〔地址〕、〔数据类型〕、〔注释〕。

一个典型的行的含义是：为输入的〔地址〕定义一个符号变量〔名称〕，并指定其〔数据类型〕，也可以为这个符号变量写一些〔注释〕，当然，〔注释〕是可有可无的。比如，图 4-32 的第 1 行的含义是：为地址“%V20.0”定义一个符号变量，变量名称为“PreviousT20”，其数据类型为“BOOL”。符号变量的命名规则请参见 3.4.1 标识符的定义 部分。

- 〔功能块实例〕页面

用于定义功能块的实例。样式如下图所示。

VAR_GLOBAL

序号	FB实例名称	FB类型	注释
1	T11	TOF	
2	T10	TON	
3			
4			
5			
6			
7			
8			

变量 / 功能块实例

图 4-33 全局变量表中的〔功能块实例〕页面

在 3.6.5 FB 实例的命名及使用 提到过，为了方便用户的使用，功能块实例的定义是由 EasyProg 自动完成的，因此在〔功能块实例〕页面中，〔FB 实例名称〕以及〔FB 类型〕是不可编辑的，用户仅可以在〔注释〕部分输入对于该实例的有关说明。由于功能块实例定义区的操作非常简单，因此下面将略过不予介绍。

4.9.1 如何进入全局变量表窗口?

有如下三种方式可以进入全局变量表窗口：

- 双击工程管理器中【资源】组下的〔全局变量表〕节点；
- 在〔全局变量表〕节点上单击鼠标右键，然后执行弹出的〔打开…〕菜单命令。
- 执行【查看】→〔全局变量表〕菜单命令。

4.9.2 在全局变量表窗口中进行编辑

- 输入数据

鼠标单击某表格单元，即可让该单元进入编辑状态。

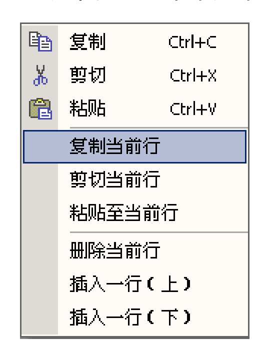
敲任意键也可以让具有焦点的表格单元进入编辑状态。

输入结束后，须敲 Enter 键，或者鼠标单击输入单元之外的表格任意地方来进行确认。

若输入的数据有错误，将会自动变成红色进行提示。错误的数据在保存时会被略掉。

- 右键菜单

在表中任意一个单元单击鼠标右键，将会弹出如下菜单：



- 〔复制〕：将具有焦点的表格单元的内容复制至剪贴板中，快捷键 Ctrl+C。
- 〔剪切〕：删除具有焦点的表格单元的内容，并将其复制至剪贴板中，快捷键 Ctrl+X。
- 〔粘贴〕：将剪贴板中的内容复制至具有焦点的表格单元中，快捷键 Ctrl+V。
- 〔复制当前行〕：复制焦点所在行中各表格单元的内容。
- 〔剪切当前行〕：剪切焦点所在行中各表格单元的内容。
- 〔粘贴当前行〕：将剪贴板中内容复制至焦点所在行的各表格单元中。
- 〔删除当前行〕：删除焦点所在的行。
- 〔插入一行（上）〕：在焦点所在行的上一行的位置插入一个新的空行。
- 〔插入一行（下）〕：在焦点所在行的下一行的位置插入一个新的空行。

4.10 交叉索引表

交叉索引表包含如下部分：

- 〔交叉索引〕页面

用于分析当前工程中用到的所有地址变量并列表显示详细信息。

〔交叉索引〕页面如图 4-34 所示。

交叉索引表

地址	全局变量名	POU	位置	读/写
%M10.0		GL_JK	Network 0	写
%M10.0		GL_JK	Network 1	写
%SM0.0		GL_JK	Network 0	读
%SM0.0		GL_JK	Network 1	读

交叉索引

图 4-34 交叉索引表中的〔交叉索引〕页面

- 〔地址〕　　：显示了当前工程中用到的所有直接地址。
- 〔全局变量名〕：显示本行〔地址〕所对应的全局变量名。
- 〔POU〕　　：显示本行〔地址〕所在的 POU 名称。
- 〔位置〕　　：指明本行〔地址〕在〔POU〕中的具体位置。

  若 POU 用 IL 编写，则显示的是行号；若用 LD 编写，则显示的是网络号。
- 〔读/写〕　　：指明本行〔地址〕在所处的〔位置〕上被进行了读或者是写操作。

如图 4-34，表格中第一行的意思是：在本工程 *GL_JK* 程序的 *Network 0* 中用到了 一次 *%M10.0*，此处对 *%M10.0* 进行的是 *写* 操作，*%M10.0* 没有被定义全局变量名称。

交叉索引表中的信息在在第一次编译之后才能生成，并在以后的编译过程中自动刷新。

4.10.1　如何进入交叉索引表?

有如下三种方式可以进入全局变量表窗口：

- 执行【查看】→〔交叉索引表〕菜单命令；
- 鼠标单击工具栏上的图标。

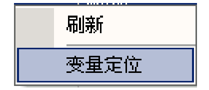
4.10.2　在交叉索引表中进行操作

- 对表格的操作

鼠标双击某一行，即可打开本行的〔POU〕并定位到〔地址〕所在的具体〔位置〕上。

- 右键菜单

在表格任何一行上单击鼠标右键，将弹出如下右键菜单：



► 〔地　　址〕：刷新显示交叉索引数据。

► 〔变量定位〕：打开本行的〔POU〕并定位到〔地址〕所在的具体〔位置〕上。

4.11 变量状态表

用户可以利用变量状态表来对当前工程中用到的任意地址变量进行在线监测和强制。变量状态表的样式如下图所示。

变量状态表

序号	地址	变量名	格式	当前值	强制值
1	%I1.2	test	BOOL	0	0
2	%I2.3		BOOL	0	0
3	%Q1.2		BOOL	0	0
4	%VW100		有符号数	0	
5					

图 4-35 变量状态表

表中共分如下五列：

► 〔地　址〕：输入将要被监测和强制的直接地址。

► 〔变量名〕：显示本行〔地址〕所对应的全局变量名。

► 〔格　式〕：选择当前值和强制值的数据显示格式，

可选的格式有 BOO、REAL、有符号数、无符号数、二进制、16 进制等。

► 〔当前值〕：在线监测时将显示监测到的本行〔地址〕的值。

► 〔强制值〕：在线监测时可以在此输入本行〔地址〕将要被强制为的值。



提示：

为了提高效率，EasyProg 只允许对当前工程中用到的变量进行监测和强制。若用户输入了未被用到的变量，EasyProg 虽然不提示错误，但当前值、强制值均不会起作用。

4.11.1 如何进入变量状态表窗口?

有如下三种方式可以进入变量状态表窗口:

- 双击工程管理器中【资源】组下的〔变量状态表〕节点;
- 在〔变量状态表〕节点上单击鼠标右键,然后执行弹出的〔打开…〕菜单命令;
- 执行【查看】→〔变量状态表〕菜单命令。

4.11.2 在变量状态表中进行编辑

- 输入数据

鼠标单击某允许编辑的表格单元即可让该单元进入编辑状态。

敲任意键也可以让具有焦点的表格单元进入编辑状态。

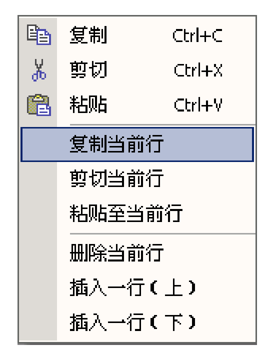
输入结束后,须敲 Enter 键,或者鼠标单击输入单元之外的表格任意地方来进行确认。

若输入的数据有错误,将会自动变成红色进行提示。错误的数据在保存时会被略掉。

用户在输入〔强制值〕时,可以输入任何合法的常量,软件会根据所选的〔显示格式〕自动对用户输入的数据进行格式的转换。

- 右键菜单

在表中任意一个单元单击鼠标右键,将会弹出如下菜单:



► 〔复制〕:将具有焦点的表格单元的内容复制至剪贴板中,快捷键 Ctrl+C。

- 〔剪切〕：删除具有焦点的表格单元的内容，并将其复制至剪贴板中，快捷键 Ctrl+X。
- 〔粘贴〕：将剪贴板中的内容复制至具有焦点的表格单元中，快捷键 Ctrl+V。
- 〔复制当前行〕：复制焦点所在行中各表格单元的内容。
- 〔剪切当前行〕：剪切焦点所在行中各表格单元的内容。
- 〔粘贴当前行〕：将剪贴板中内容复制至焦点所在行的各表格单元中。
- 〔删除当前行〕：删除焦点所在的行。
- 〔插入一行（上）〕：在焦点所在行的上一行的位置插入一个新的空行。
- 〔插入一行（下）〕：在焦点所在行的下一行的位置插入一个新的空行。

4.11.3 如何利用变量状态表强制变量?

① 进入变量状态表，输入有效的地址、强制值；

② 与 PLC 联机；

③ 执行【调试】→〔变量强制〕菜单命令，或者单击工具栏上的图标，即可将变量状态表中的所有强制值写入相应的地址。

④ 执行【调试】→〔取消全部强制〕菜单命令，或者单击工具栏上的图标，即可取消全部变量的强制状态。



注意：

KDN-K3 系列 CPU 采用强制优先模式！一定要记得在调试结束后取消强制状态!

第五章 使用 EasyProg 创建用户程序

本章对 EasyProg 中 LD 和 IL 编辑器的功能、使用进行了详细的描述，同时也阐述了 IEC61131-3 标准中关于 LD、IL 语言的相关语法、规定。

通过阅读本章，用户可以轻松地使用 EasyProg 编写出符合 IEC 标准的应用程序。

针对 PLC 应用程序的编写，IEC61131-3 中规定了 3 种文本化语言和 3 种图形化语言。文本化语言包括：指令表（IL）、结构化文本（ST）、顺序功能图（SFC，文本化版本）；图形化语言包括：梯形图（LD）、功能块图（FBD）、顺序功能图（SFC，图形化版本）。

EasyProg 目前支持 IL 语言和 LD 语言编程。在一个工程中，IL 和 LD 语言可以混用，但是在一个 POU 中就只允许使用一种语言。

5.1 EasyProg 中的 POU 类型

在 3.7.1 工程的组织结构 部分中已经介绍了在 EasyProg 中共支持三类 POU：主程序、子程序、中断服务程序。主程序、子程序、中断服务程序均为 IEC61131-3 中规定的 PROGRAMME 类型，主程序、中断服务程序可以作为任务在 CPU 中运行。

5.1.1 主程序

主程序是 CPU 的主循环任务，在 CPU 的每次循环中均扫描、执行主程序一次。

请参阅前文 3.8 CPU 中程序的执行 部分。

主程序没有参数，但允许使用局部变量。

5.1.2　子程序

子程序是可重用的代码段，用户可以将常用的控制功能编写成一个个子程序。使用子程序可以更好地组织应用程序的结构，方便调试和维护，有效地缩短程序代码的长度。子程序必须被主程序或者中断服务程序调用后才能被执行。

在子程序中可以使用局部变量，也可以有形式参数。在每次调用带参数的子程序时可以将不同的实际变量赋给形式参数，从而能够对不同的变量、数据进行相同的处理运算。

参数、局部变量的命名规则请参见3.4.1 标识符的定义，允许使用如下几种变量类型：

- VAR：局部变量。
- VAR_INPUT：输入参数。
- VAR_OUTPUT：输出参数。
- VAR_IN_OUT：输入、输出参数。

5.1.3　中断服务程序

采用中断技术的目的是提高 CPU 的执行效率，实现对内部或者外部各种预定义中断事件的快速响应。KDN-K3 系列 PLC 定义了数十种中断事件，每一种中断都被指定了唯一的事件号以供 CPU 识别。中断事件各有不同的中断优先级。当中断事件发生时会按照优先级和发生的时序进行排队，队列中优先级高的中断事件首先得到处理，优先级相同的中断按照发生的时序进行处理。

响应某个中断事件而被执行的程序称为中断服务程序。在工程中用户建立中断服务程序，并通过中断指令将其与某个中断事件号联系起来，则当该事件号对应的中断事件发生时，CPU 将自动调用中断服务程序进行处理。由于有快速响应的要求，因此用户应当尽量优化中断服务程序，使其短小精干。关于中断指令的描述请参阅后面的指令集。

一个中断事件只能对应一个中断服务程序，但一个中断服务程序可以对应多个中断事件。

中断服务程序没有参数，但允许使用局部变量。

5.1.3.1　中断事件分类

在 KDN-K3 系列 PLC 中，中断事件分为了如下三类：

- 通讯口中断

用于自由口通讯。这一类中断的优先级最高

- I/O 中断

包含了上升沿或下降沿中断、高速计数器中断和脉冲输出（PTO）中断。这一类中断的优先级中等。

上升沿、下降沿中断只能由 CPU 本体集成 DI 的第 0 至 3 通道（地址为 I0.0---I0.3）捕获。

PTO 中断是在指定数目的脉冲输出完成时发生。它的一个典型应用是步进电机的控制。

- 时间中断

包含了定时中断和定时器中断。这一类中断的优先级最低。

定时中断周期性的产生，可以利用它来完成周期性的任务。

5.1.3.2　中断事件列表

事件号	中断描述	分类	优先级
32	PORT 1：发送数据完成	通讯中断 (最高优先级)	最高
31	PORT 1：接收数据完成		↓
30	PORT 0：发送数据完成		
29	PORT 0：接收数据完成		
28	PTO 0 完成	I/O 中断 (中等优先级)	
27	PTO 1 完成		
26	I0.0 检测到下降沿		

25	I0.0 检测到上升沿		
24	I0.1 检测到下降沿		
23	I0.1 检测到上升沿		
22	I0.2 检测到下降沿		
21	I0.2 检测到上升沿		
20	I0.3 检测到下降沿		
19	I0.3 检测到上升沿		
18	HSC0　CV=PV（当前值=预设值）		↓
17	HSC0　输入方向改变		
16	HSC0　外部复位		
15	HSC1　CV=PV（当前值=预设值）		
14	HSC1　输入方向改变		
13	HSC1　外部复位		
12	HSC2　CV=PV（当前值=预设值）		
11	HSC2　输入方向改变		
10	HSC2　外部复位		
9	HSC3　CV=PV（当前值=预设值）		
8	HSC4　CV=PV（当前值=预设值）		
7	HSC4　输入方向改变		
6	HSC4　外部复位		
5	HSC5　CV=PV（当前值=预设值）		最低

表 5-1　KDN-K3 系列 PLC 支持的中断事件

5.2 IL 编程

5.2.1 IL 的背景

IL 语言是一种低级语言，与汇编语言非常相似，是在借鉴、吸收世界上各著名 PLC 厂商的指令表语言的基础上而形成的一种标准语言。

在其它语言进行编译时，IL 是作为其中的公用中间语言。因此通常情况下，与其它语言相比，使用 IL 编写的程序效率更高、更加紧凑。

5.2.2 IL 的语法规定

5.2.2.1 IL 语句格式

IL 程序面向行，每行语句只能有一条指令。IL 程序中允许有空白行存在。

IL 程序中一个语句的基本格式如下图：

```
标号：
操作符/功能/功能块    操作数（表）    (* 注释 *)
```

图 5-1 IL 语句格式

- 〔标号〕

  可选。使用标号的目的就在于实现程序的跳转。标号的命名格式如同变量。

- 〔操作符/功能/功能块〕

  请参见下一章中对于各操作符、功能、功能块的详细说明。

- 〔操作数表〕

  在操作符、功能、功能块的说明中包含了对应的操作数的描述。

  在操作符、操作数之间至少有一个空格。

- 〔注释〕

可选。在所有的语言中，注释的格式都是相同的，由（*　*）进行界定。

每行只允许有一个注释，注释不允许嵌套，嵌套的注释编译器将认为是错误。

举例如下：

```
(* NETWORK 0 *)
Run:                (* 语句标号，供跳转时使用 *)
LD      %I1.0
TP      T2, 168     (* 若 I1.0 = true，启动定时器 T2，T2 被声明为 TP 型 *)
```

5.2.2.2　CR 值

IL 中提供了一个“当前结果”（CR，Current Result）累加器，在 CR 中存储了用户程序的当前执行结果。用户程序中的每一行语句执行后，CR 值都会被“刷新”。CR 值可以是编程软件支持的任意数据类型，这取决于刚刚执行完的语句。依据后续语句的不同，CR 值可能作为下一条语句的执行条件，也可能作为下一条语句的操作数之一。

操作符不同，执行之后对 CR 值的影响也不同。下表根据对 CR 值不同的影响将 EasyProg 中的操作符进行了初步的分组。更详细的描述请参见下一章对于各指令的介绍。

操作符	**分组缩写**	**对 CR 值的影响**
LD，LDN	C	CR 值重新建立
逻辑指令，比较指令等	P	CR 值被更新为操作结果
ST，R，S，JMP，JMPCN，JMPC 等	U	CR 值保持不变

表 5-2　各操作符执行之后对于 CR 值的影响

在 IEC61131－3 中并没有完善地定义各种操作符对于 CR 的影响，因此在不同的编程系统中这些定义可能有所不同。

5.2.2.3 网络

与其它语言相同，IL 编写的 POU 包括如下部分：

- POU 的起始和结束指令。编程软件可以要求用户输入这一部分，也可以将其自动生成。
- 参数、变量声明部分；
- 代码部分。

在 IL 程序中也存在着网络（Network）的概念，以网络作为基本的段落，用户程序的代码部分就是由若干个网络组成。一个典型的网络包括：

- 网络标号；
- 网络中的语句。

5.2.3 在 EasyProg 中使用 IL 编程

当建立一个新程序时，若选择其语言为“指令表（IL）”，就将进入 IL 编辑器进行编程；若打开一个用 IL 编写的程序，也将进入 IL 编辑器。IL 编辑器的外观如下图。

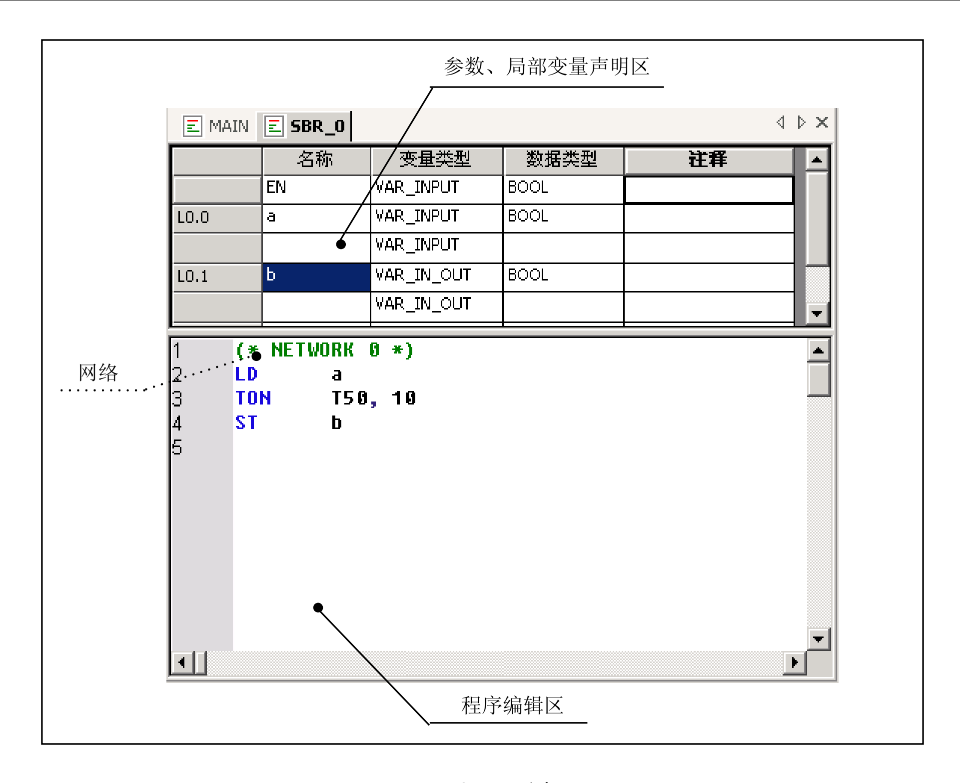


图 5-2 IL 编辑器

编辑器分为两部分区域：

- 参数、局部变量声明区：用于声明 POU 的参数和 POU 内用到的局部变量。支持右键菜单。
- 程序编辑区：用户在此编写自己的应用程序。

5.2.3.1 添加一个网络

在 IL 程序中添加一个网络的方法如下：

- 执行 【IL/LD】→〔IL 加入网络〕 菜单命令；
- 单击工具栏上的 图标；
- 单击鼠标右键，执行弹出的 〔IL 加入网络〕 菜单命令。

5.2.3.2 网络中允许的格式

① 在一个网络中可以只有一个语句标号。例如：

```
(* NETWORK 0 *)
MRun:      (* 可以只有一个标号 *)
```

② 在一个网络中可以只有程序语句。

在 5.2.2.2 CR 值中，我们将操作符分了“C”、“P”、“U”几类。

网络中的程序必须以“C”组指令开始。

网络中的程序必须以“P”、“U“组指令或者功能、功能块的调用结束。

举例如下：

```
(* NETWORK 0 *)
LD   %M3.5   (* 以 LD 指令开始 *)
… …          (* 用户可以输入其它指令 *)
ST   %Q2.3   (* 以允许作为结尾的指令结束 *)
```

③ 在一个网络中可以有语句标号和程序语句。

程序语句的起始、结束必须遵循在②中的原则。

举例如下：

```
(* NETWORK 0 *)
MRun:
LD %M3.5   (* 以 LD 指令开始 *)
… …        (* 用户可以输入其它指令 *)
ST %Q2.3   (* 以允许作为结尾的指令结束 *)
```

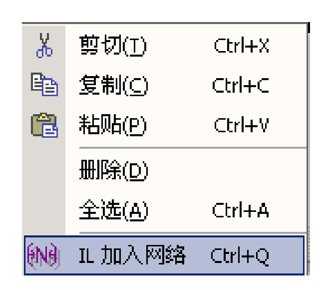
5.2.3.3　IL 编辑器中的操作

IL 编辑器能自动地对用户输入的语句进行格式化。若输入的语句有错误，则会在该行的开头显示一个红色的问号（? ）。

IL 编辑器类似于一个文本编辑器，支持通常的键盘操作，比如删除(Del)、退格(BackSpace)、光标移动、移动光标时同时按下 Shift 键可以选中文本等。

在 IL 编辑器中可以使用【编辑】菜单中的所有编辑命令。

在程序编辑区中单击鼠标右键，将会弹出如下右键菜单：



这些菜单命令在前面已经介绍过，在此不再详述。

5.2.3.4　在线调试

若当前窗口是 IL 编辑器，则执行【调试】→〔在线监测〕菜单命令可让其进入在线监测状态。

若当前已经处于线监测状态，将当前窗口切换至 IL 编辑器窗口，则该 IL 窗口也将进入在线监测状态。在线监测状态下是不允许进行编辑的。

在线监测状态下，在原程序编辑区中将分为两栏，右边显示程序，左边显示相应的变量值，中间以一条竖线分割开。将光标置于竖线上，待其形状变为 ↔ 后，即可拖动该线改变左、右区域的大小。

5.2.3.5 IL 程序示例

下面这段程序让 T0、T1 相互循环启动，目的是在 T0 的输出端输出一周期为 2 秒的方波。

```
(* NETWORK 0 *)
LDN     %M0.0
TON     T0, 1000        (* 依靠 T1 的输出来启动 T0，定时为 1000×1ms *)
ST      %M0.1
LD      %M0.1
TON     T1, 1000        (* 依靠 T0 的输出来启动 T1，定时为 1000×1ms *)
ST      %M0.0

LD      %M0.1
ST      %Q0.0           (* 在 Q0.0 输出方波 *)
```

5.3 LD 编程

5.3.1 LD 的背景

LD（梯形图）语言是 IEC61131-3 标准中规定的五种编程语言之一，是 PLC 编程中被最广泛使用的一种图形化语言。

LD 语言源于机电一体化领域中图形表示的继电器逻辑，用它编写的程序能够与实际的电气操作原理图相对应，非常直观，易于学习和掌握。LD 语言的长处在于布尔逻辑的处理。下图是用 LD 编写的一段简单程序。

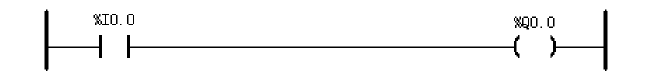


图 5-3 LD 程序示例

5.3.2　LD 的网络结构

如同 IL 一样，在 LD 中也使用“网络（Network）”的概念，以网络作为基本的段落。

一个 LD 网络的边界是位于左、右两侧的电源线（Power Rails）。左侧的电源线相当于“火线”，右侧的电源线相当于“零线”。

我们可以对 LD 程序作如下形象的描述：电能自左侧的“火线”沿着连接线流经线上所有的元件（包括触点、线圈、功能、功能块等），这些元件依据自身的逻辑状态，或是中断能量流，或是将电能传输到后继的元件并最终到达右侧的“零线”。

5.3.3　LD 的图形对象

LD 网络中可以包括如下图形对象：

①　连接

LD 中使用水平连接线和垂直连接线，分别对应着串联和并联的关系。下表提到的连接线的值或者连接线某侧的值也可以理解为是否有能量流存在。

图形对象	名 称	描 述
	水平连接	可以将其理解为一根理想的导线。 在水平连接线上任一点处的值（true 或 false）都是相同的。
	垂直连接 （含有水平连接）	垂直连接线左侧的值被传递到右侧所有的水平连接上。
		垂直连接线左侧是并联的水平连接线。 左侧所有水平连接的值首先进行“或”运算，然后再将运算结果传递到垂直连接线的右侧。
		相当于上述两种垂直连接的组合。 左侧所有水平连接的值首先进行“或”运算，然后再将运算结果传递到垂直连接线右侧所有的水平连接上。

表 5-3　LD 中的连接

② 接点

接点有两种类型：常开接点和常闭接点。

每个接点都必须对应一个有效的布尔型变量，变量名称被写在图形元素的上面，该变量的值决定了接点的状态（闭合或者断开），并进而决定了接点所在线路的通或者断。

图形对象	名 称	描 述
变量名 ─┤ ├─	常开接点	若变量的值为 TRUE，则接点闭合，线路导通； 若变量的值为 FALSE，则接点断开，线路断开。
变量名 ─┤/├─	常闭接点	若变量的值为 TRUE，则接点断开，线路断开； 若变量的值为 FALSE，则接点闭合，线路导通。

表 5-4　LD 中的接点

③ 线圈

线圈是用于给布尔型变量赋值的图形元素。

图形对象	名 称	描 述
变量名 —( )—	线圈	将左连接的值传递至变量
变量名 —(S)—	置位线圈	若左连接的值为 true，则将变量置为 true； 若左连接的值为 false，则变量值保持不变。
变量名 —(R)—	复位线圈	若左连接的值为 true，则将变量置为 false； 若左连接的值为 false，则变量值保持不变。

表 5-5　LD 中的线圈

④ 程序执行顺序控制

为了定义程序的执行顺序，在 LD 中提供了如下两种类型的控制执行顺序的元素：

图形对象	名 称	描 述
├── (1) ——<RETURN>	条件返回	若 RETURN 元素左边连接的值为 true，则脱离当前 POU 并返回到调用 POU 中；否则将其略过。
├──>> 语句标号	无条件跳转	无条件跳转到指定的“语句标号”处。
├── (1) ——>> 语句标号	条件跳转	若元素左连接的值为 TRUE，则跳转到指定的“语句标号”处；否则将其略过。

表 5-6　LD 中的执行顺序控制元素



注：(1)　表示此处存在结果为布尔型的图形代码。

⑤　调用功能和功能块

LD 支持对功能和功能块的调用。被调用的功能和功能块以矩形框来表示，其形参显示在矩形框内，实参显示在矩形框外部的连接线上。功能块的参数中至少有一个 BOOL 型的输入参数和一个 BOOL 型的输出参数，用于将功能块接至连接线上。

功能必须有一个名为 EN 的 BOOL 型输入和一个名为 ENO 的 BOOL 型输出，用于控制该功能的执行。若 EN 的值为 1，则该功能被执行且 ENO 也将被置为 1；若 EN 的值为 0，则不执行该功能且 ENO 也将被置为 0。

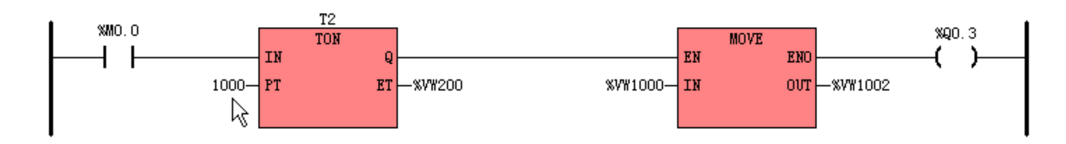


图 5-4　功能块和功能调用的示例

5.3.4　在 EasyProg 中使用 LD 编程

当建立一个新程序时，若选择其语言为“**梯形图（LD）**”，就将进入 LD 编辑器进行编程；若打开一个用 LD 编写的程序，也将进入 LD 编辑器。LD 编辑器的外观如下图。

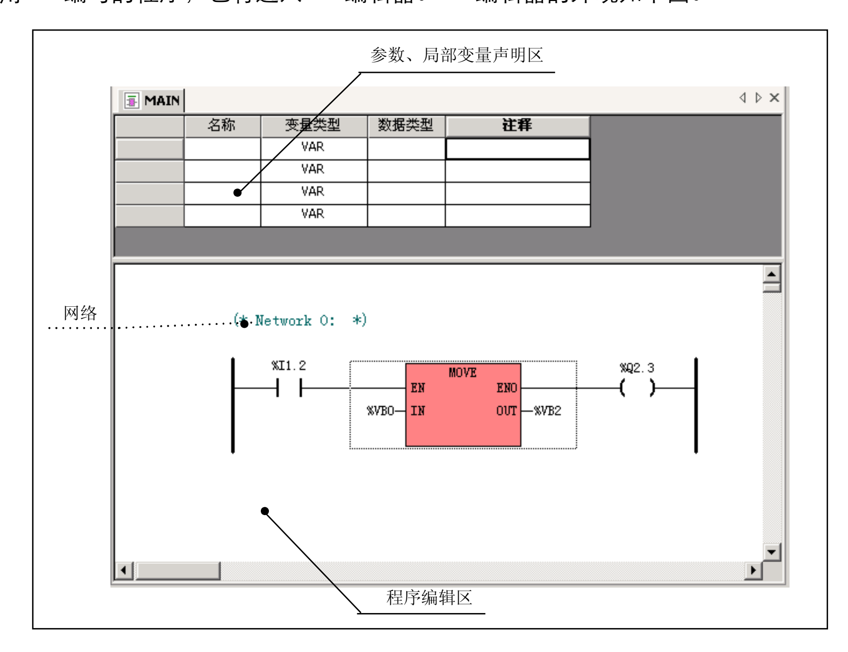


图 5-5　LD 编辑器

5.3.4.1　LD 程序的限制

在每一个 LD 程序中，最多不超过 100 个网络。

在 LD 程序中，每条连接线路径上的元件数量限制：若只有线圈、触点，则建议不超过 31 个触点加 1 个线圈；若只有功能/功能快，则建议不超过 12 个块加 1 个线圈加 1 个触点。

在 LD 程序中一个并联不允许超过 16 个分支，不允许出现单独两个功能/功能块并联的情况。

5.3.4.2　使用鼠标和键盘进行编辑操作

鼠标左键单击某元件可将该元件选中并将焦点置于其上（为该元件“套”上一个矩形框）；双

击某元件可以打开其属性对话框修改属性；鼠标右键单击某元件会弹出如下右键菜单：



所有的菜单命令均已在前面描述过，此处不再赘述。

在 LD 编辑器内也支持键盘操作：使用键盘上的上、下、左、右方向键可以移动焦点；敲 Enter 键，则可以打开具有焦点的元件的属性对话框以修改其属性；敲 Delete 键可以删除具有焦点的元件；另外，各菜单命令都有对应的快捷键，使用快捷键与使用菜单命令等效。

5.3.4.3 LD 编程步骤

下面的描述将以鼠标操作为主，至于键盘操作方式请参阅上一小节的介绍。

① 首先利用菜单、工具栏或者右键菜单来执行“LD 加入网络”命令以加入一个网络（network），如图 5-6。双击网络标号部分，即可弹出对话框，用户可以输入该网络的注释。

(* Network 0: *)

????

(END)

图 5-6 LD 编程第一步：加入网络

② 双击接点或者线圈将弹出如下对话框，可修改该元件的类型、连接的变量：

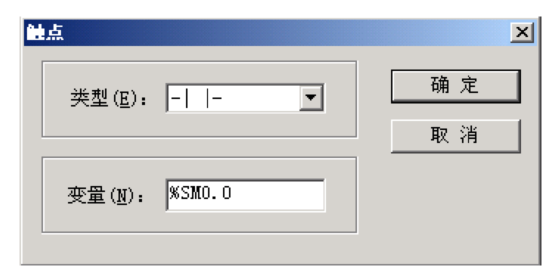


图 5-7　LD 编程：修改接点或者线圈的属性

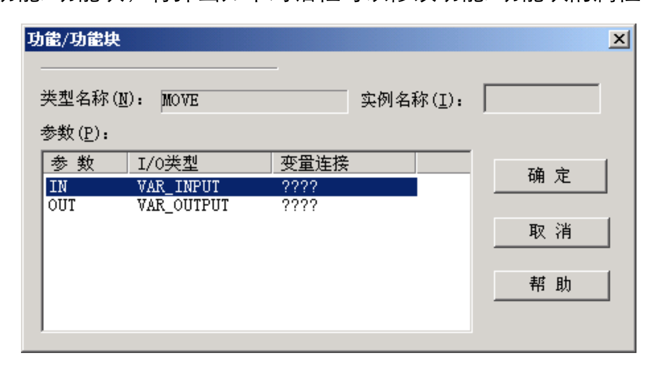
③　单击某元件将其选中作为基准，可以继续执行菜单、工具栏命令或快捷键加入其它元件；选好基准后，也可以双击右侧“**LD 指令集**”中的相应指令将其加入；或者右键单击某元件，执行相应的弹出菜单命令也可。 下面是加入一个 MOVE 功能之后的图形。

(* Network 0:　*)

%I0.0　MOVE　EN　ENO　(END)　????　IN　OUT　????

图 5-8　LD 编程：继续加入其它的元件

④　双击程序中的功能/功能块，将弹出如下对话框可以修改功能/功能块的属性：



双击〔**参数**（P)〕中的任何一项参数，即可修改该参数连接的变量。

或者使用上、下方向键选择某项参数，敲 Enter 键，也可修改该参数连接的变量。使用 Tab 键可以在各个按钮、列表之间切换焦点。

软件将对用户的输入进行严格的语法检查，并将对错误进行提示。



图 5-9 LD 编程：继续修改其它元件的属性

修改完后的网络如下图：

(* Network 0: *)

%I0.0
MOVE
EN
ENO
%AIW0
IN
OUT
%VW0
END

图 5-10 LD 编程：修改完的 Network 0

⑤ 完成当前网络的编写后，继续执行“LD 加入网络”的命令加入新的网络：若选中某网络的标号部分作为基准，将在该网络的前面加入新网络；若选中某网络中元件作为基准，将在该网络的后面加入新网络。

⑥ 继续执行其它命令，包括使用菜单命令、工具栏命令、快捷键、右键菜单命令或者利用右侧的“LD 指令集”，直到完成 LD 程序的编写。

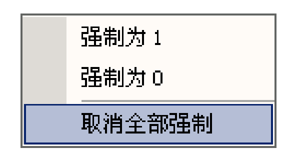
5.2.3.5 在线调试

若当前窗口是 LD 窗口，则执行【调试】→〔在线监测〕菜单命令可让其进入在线监测状态。若当前已经处于在线监测状态，然后将当前窗口切换至某 LD 编辑器窗口，则该 LD 窗口也将

进入在线监测状态。在线监测状态下，各种模拟量数据的显示可执行【工具】→〔软件设置…〕菜单命令进行修改。

在线监测状态下是不允许进行编辑的。

在线监测状态下，在接点或者线圈上单击鼠标右键，将弹出如下右键菜单：



- ▶ 〔强制为 1〕：将当前接点或者线圈连接的变量强制为 TRUE。
- ▶ 〔强制为 0〕：将当前接点或者线圈连接的变量强制为 FALSE。
- ▶ 〔取消全部强制〕：取消全部变量的强制状态。

5.2.3.6　LD 程序示例

(* Network 0: 下面这段程序让T0、T1相互循环启动，
目的是在T0的输出端输出一周期为2秒的方波。 *)

%M0.0　T0　TON　IN　Q　1000　PT　ET　%VW0　%M0.1

(* Network 1: *)

%M0.1　T1　TON　IN　Q　1000　PT　ET　%VW2　%M0.0

(* Network 2: *)

%M0.1　%Q0.0

图 5-11　LD 程序示例

5.4 举例：开发一个用户工程的步骤

上面描述了 EasyProg 环境以及基本的使用方式、规则等，下面我们将一步步地举例说明如何创建、调试一个具体的工程。下面的描述是建立在假定用户已准备好所有的硬件的基础上的。

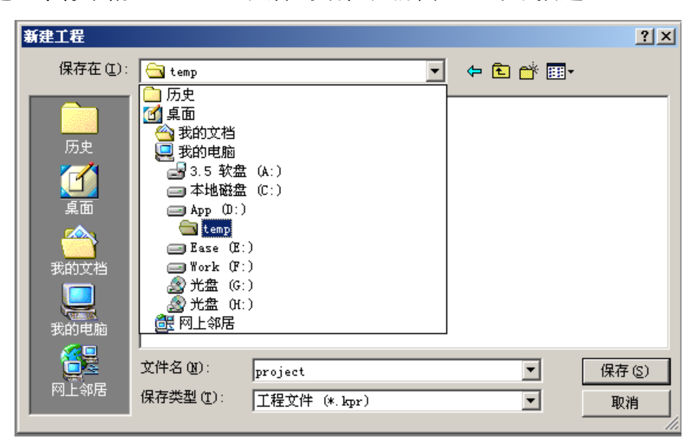
下面我们要编写一个名为 *project* 的工程，使用的硬件是一个 *K306-24DT*，工程中包含主程序和一个子程序：子程序名为 CtrlMotor，用 LD 语言编写，目的是循环启动 Q0.0—Q0.7；主程序名为 Main，用 IL 语言编写，目标是调用 CtrlMotor。在本演示中没有用到初始化数据表、全局变量表等，关于它们的使用请参阅前面章节的介绍。

① 启动 EasyProg

执行【开始】-> 【程序】-> 【KDN】-> 【EasyProgV*xxxx*】命令（*xxxx* 代表版本号），或者双击桌面上的 EasyProg 软件图标，即可进入 EasyProg 软件界面。

② 建立一个新工程

执行【文件】->【新建工程..】菜单命令，或者单击工具栏上的图标，将会出现“新建工程”对话框，这是一个标准的 Windows 文件对话框，前面已经详细描述过。



用户在此选择或者新建一个文件夹来作为工程文件的存放目录，可以为工程命名。

目录、工程名选定以后，单击【保存】按钮，创建新工程。

本例中我们选择 D:\temp 作为工程存放目录，工程名为 project。

③ 进行硬件配置

当然，用户可以在任何时候配置硬件。不过，我们建议用户在新工程中首先完成这一步。

在建立一个新工程时，软件会自动加入一个 K306-24DT，用户可根据实际情况进行修改。

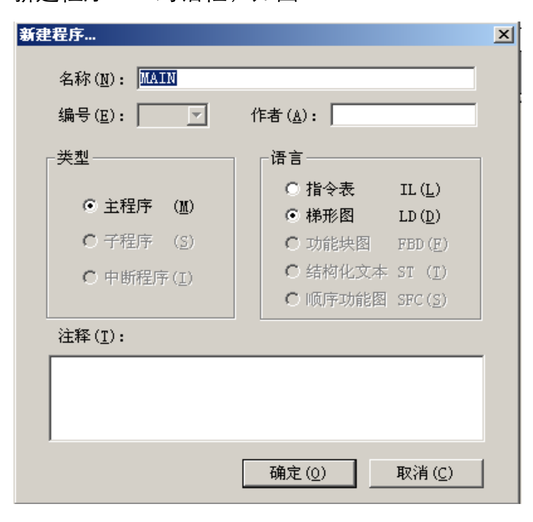
鼠标右键单击工程管理器中的【PLC 硬件配置】节点，执行【打开..】命令；或者直接双击【PLC 硬件配置】，都可以进入硬件配置环境。

④ 编程

现在就开始编程吧。

我们将以 IL 语言编写一个主程序 Main，以 LD 语言编写一个子程序 *CtrlMotor*。

鼠标右键单击工程管理器中的【程序】节点，执行【新建..】命令；或者直接单击工具栏上的图标，都可以打开“新建程序..”对话框，如图：



用户可以在此修改程序的名字，选择类型、语言，也可以注明作者、写点注释等等。

1）建立主程序

在一个新工程中，用户执行【新建..】程序命令，软件会要求首先创建主程序。主程序创建

后，才可以继续创建子程序、中断服务程序。

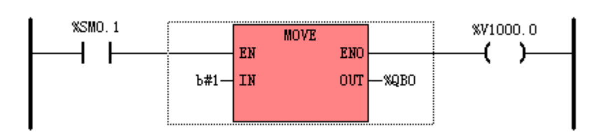
执行新建程序的命令后，在“新建程序..”对话框中，修改程序〔名称〕为 *Main* ，〔语言〕设置为“*指令表 IL*”。所有的输入完成后，单击〔确定〕按钮就创建了主程序，进入了 IL 编辑器。

由于在主程序中只需要调用子程序 *CtrlMotor*，但该子程序尚未建立，所以现在我们先不管主程序了，继续进入下一步。

2）建立子程序 *CtrlMotor*

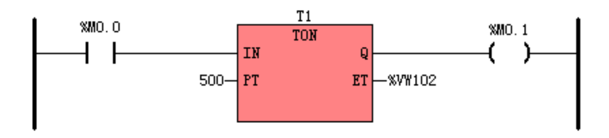
执行新建程序的命令后，在“新建程序..”对话框中，修改程序〔名称〕为 *CtrlMotor*，〔类型〕选择为“子程序”，〔语言〕选择为“梯形图 LD”。所有的输入完成后，单击〔确定〕按钮就创建了子程序 *CtrlMotor*，进入了 LD 编辑器。按照以前描述的步骤，输入以下程序：

(* Network 0: 在本网络中的目的是修改QB0的初始值。
SM0.1：当CPU执行第一次扫描时其值为1，之后其值保持为0。
利用SM0.1的特性，在CPU第一次扫描时将QB0设置为1。
当然，此处也可以使用初始化数据表的方式，请用户自己操作一下。 *)



(* Network 1: T0、T1相互循环启动，形成一个振荡电路，产生方波输出。
M0.1：周期为1s的尖脉冲；M0.0：周期为1s的方波。 *)

(* Network 2: *)



(* Network 3: 控制QB0的值循环左移。 *)

3）重新修改主程序

现在我们编完了子程序 *CtrlMotor*，需要重新返回到主程序中加入调用指令。

切换到主程序的编辑窗口，输入以下程序：

```
(* NETWORK 0 *)
LD       %SM0.0
CAL      CtrlMotor
```

⑤ 编译工程

所有工作完成后，就可以进行编译了。在编译之前，软件会自动保存工程，以确保编译的是用户最新的输入。但是，仍然建议用户注意随时保存自己的工作，以免有意外发生。

执行【PLC】->〔编译工程〕菜单命令，或者单击工具栏上的图标，就开始了编译过程。编译的结果将会在下部【信息输出窗口】中的【编译信息】页面中。如果工程中有错误，则用户需要根据信息提示对工程进行修改，直至编译成功。

⑥ 联机、下载工程

编译成功后，就可以下载工程了。在下载之前，软件会自动保存、编译工程，以确保下载的是用户最新的输入。

补充说明一下：若需要指定通讯参数，则需要先在【联机通讯设置】中完成。

执行【PLC】->【下载..】菜单命令或者单击工具栏上的图标，将当前工程下载至 CPU 中。

好，将 CPU 的运行/停止开关拨至“Run”位置，看看程序的运行情况吧。

⑦ 调试：在线监测、强制变量

在线监测命令只有在打开变量状态表、LD 程序或者 IL 程序之后方能生效。注意在线监测命令是一种复选命令，若要脱离在线状态，再执行一次在线监测命令即可。

任意打开一个刚才编写的程序，执行【调试】->【在线监测】菜单命令，或者单击工具栏上

的图标，进入在线监测，直接在程序界面上显示出了本程序内用到的直接地址的状态值。也可以进入变量状态表，在〔地址〕栏输入想要监测的地址值，进行在线监测。

进行变量强制有两种途径：在变量状态表中进行强制；或者在 LD 编辑器中执行在线状态下的右键菜单命令。具体的操作方法请参阅前面的相关章节。注意：PLC 采用的强制优先的原则，因此调试结束后一定要取消全部的强制。

第六章　KDN-K3 系列 PLC 指令集

KDN-K3 PLC 支持 IEC 61131-3 标准的基本指令及其大部分功能/功能块，编程风格符合 IEC 61131-3 标准要求，同时，根据实际的需要，对标准指令作了适当的扩充，可以满足不同用户、多应用领域工程实际的需要。

6.1 综述

本章对指令集中的所有指令进行了详细的说明，同时对大多数指令也提供了具体的使用实例。对于指令的说明包括了 LD、IL 两种格式。

在 LD 格式中，下文的描述没有具体提及能量流，能量流的含义是固定的，一个网络上各指令的状态控制着能量流在本网络上的流动。我们可以认为它是部分指令（无 EN、ENO 操作数的指令）的虚拟输入，并且其类型是 BOOL 型。为了便于描述，在下文中我们将能量流是否到达某点简称为在该点能量流的值为 1 或者 0。另外，在下文中对 EN、ENO 操作数和其数据类型也没有说明，因为它们均为 BOOL 型，且含义也是固定的：EN 的意思为“使能”，若某功能/功能块的 EN 值为 1，则该功能/功能块才被执行，能量流继续向前流动，在 ENO 端输出为 1，否则该功能/功能块不被执行，在 ENO 端输出为 0。

另外，在 5.2.2.2 CR 值 中提到，在 IL 程序中，每一条指令执行完之后都会对 CR 值产生相应的影响。本章对每条 IL 指令对 CR 值的影响都作了说明，在说明时采用了在 5.2.2.2 CR 值 中规定的“分组缩写”符号。

6.2 位指令

6.2.1 常开触点

- 指令及其操作数说明

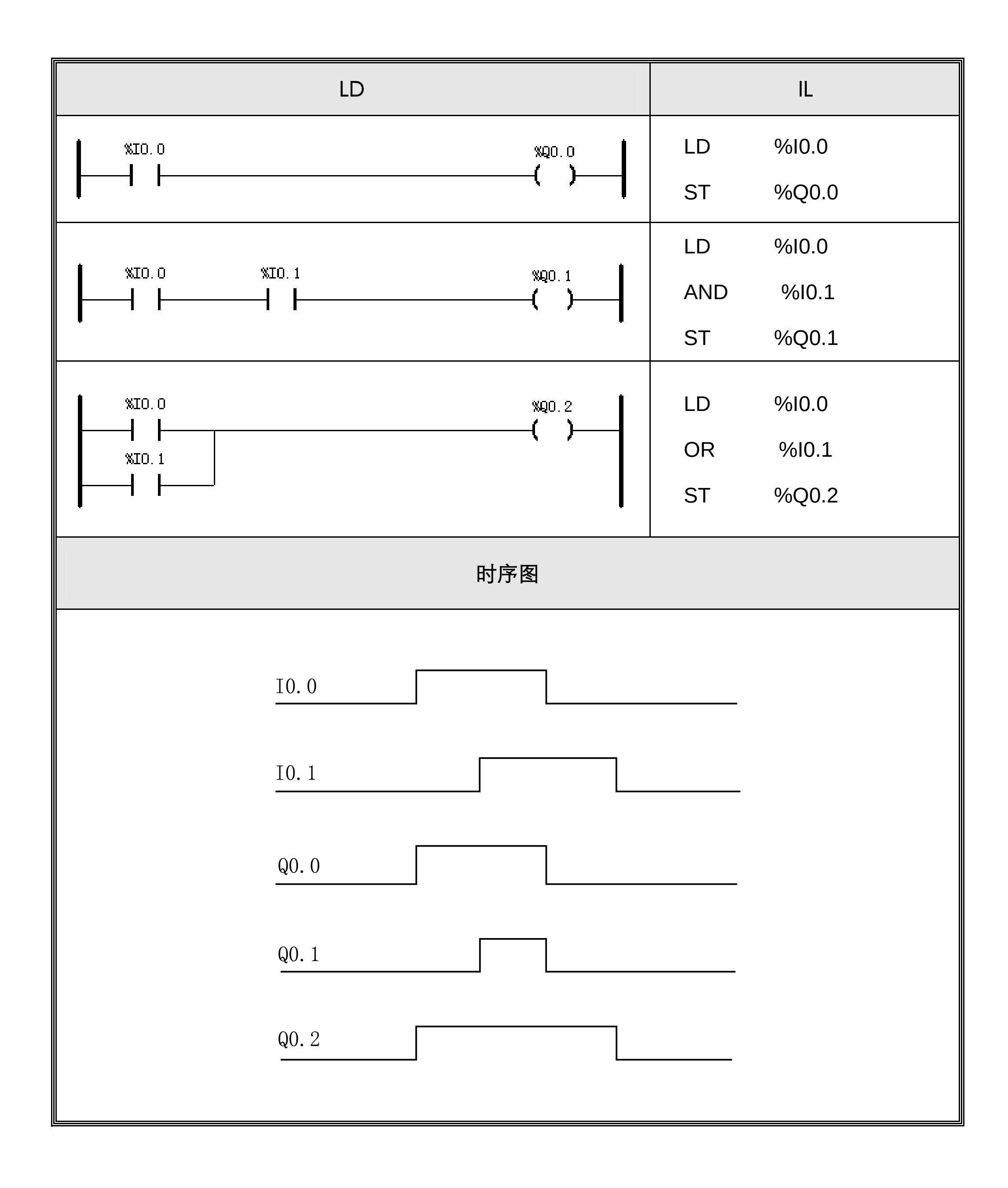
<table>
<tr><th></th><th>名称</th><th>指令格式</th><th>影响 CR 值</th><td rowspan="5">☑ CPU304
☑ CPU306
☑ CPU308</td></tr>
<tr><td>LD</td><td>常开触点</td><td>bit
─┤ ├─</td><td></td></tr>
<tr><td rowspan="3">IL</td><td>装载</td><td>LD bit</td><td>C</td></tr>
<tr><td>与</td><td>AND bit</td><td rowspan="2">P</td></tr>
<tr><td>或</td><td>OR bit</td></tr>
</table>

操作数	输入/输出	数据类型	允许的内存区
bit	输入	BOOL	I、Q、V、M、SM、L、T、C、常量

在 LD 中，当 *bit* 值为 1 时，触点闭合，能量流继续向后传递；当 *bit* 值为 0 时，触点断开，能量流也被切断。

在 IL 中，常开触点由 *LD*、*AND*、*OR* 指令提供。*LD* 指令将 *bit* 值装载于 CR 寄存器中并作为新的 CR 值；*AND* 指令将当前 CR 值和 *bit* 值进行“与”运算，并将运算结果作为新的 CR 值；*OR* 指令将当前 CR 值和 *bit* 值进行“或”运算，并将运算结果作为新的 CR 值。

• 指令使用举例



LD	IL
%I0.0 — %Q0.0	LD %I0.0 ST %Q0.0
%I0.0 — %I0.1 — %Q0.1	LD %I0.0 AND %I0.1 ST %Q0.1
%I0.0 / %I0.1 — %Q0.2	LD %I0.0 OR %I0.1 ST %Q0.2

时序图

6.2.2 常闭触点

- 指令及其操作数说明

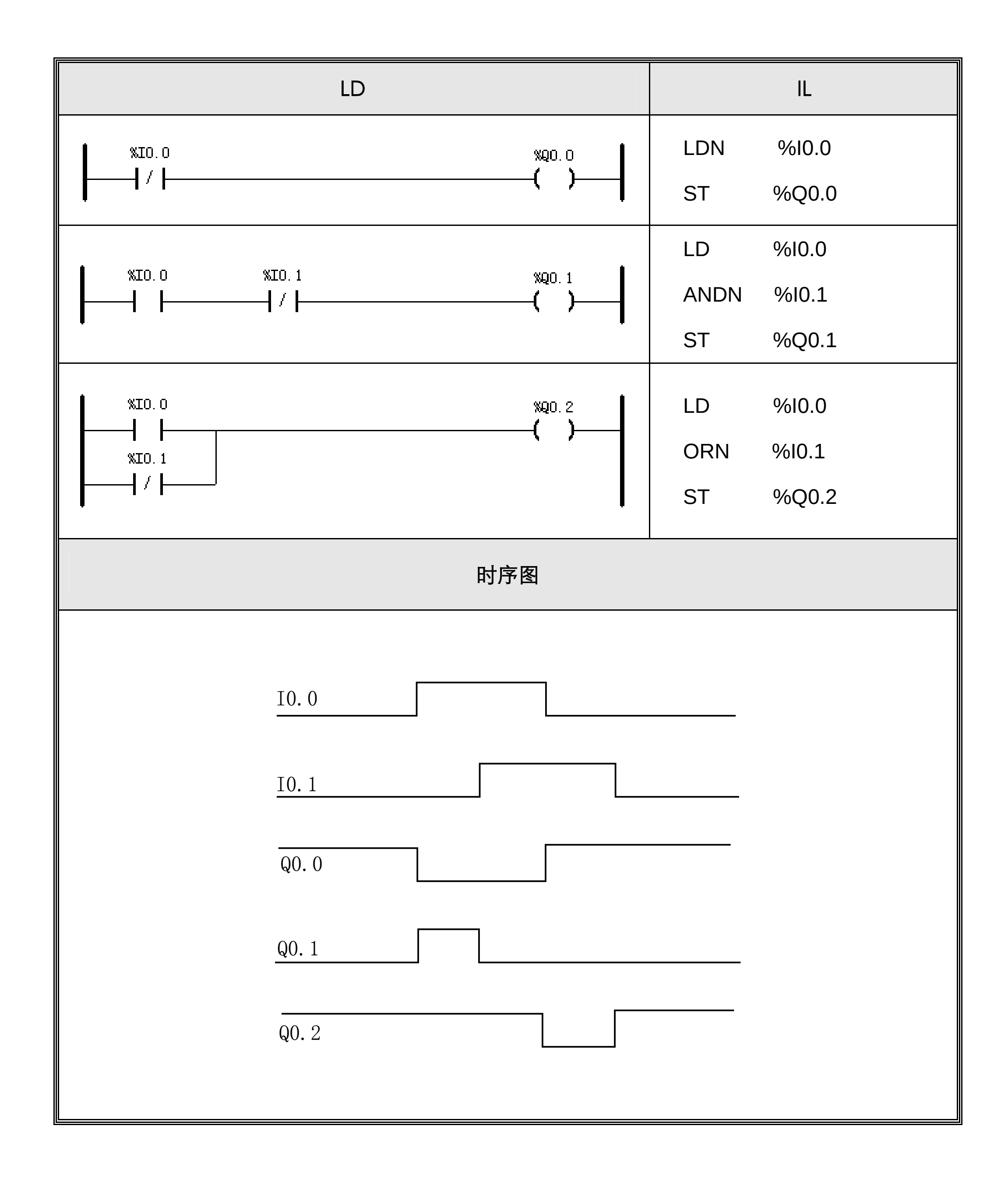
	名称	指令格式	影响 CR 值	
LD	常闭触点	bit ─┤/├─		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	取反装载	LDN bit	C	
	与非	ANDN bit	P	
	或非	ORN bit		

操作数	输入/输出	数据类型	允许的内存区
bit	输入	BOOL	I、Q、V、M、SM、L、T、C、常量

在 LD 中，当 *bit* 值为 0 时，触点闭合，能量流继续向后传递；当 *bit* 值为 1 时，触点断开，能量流也被切断。

在 IL 中，常闭触点由 *LDN*、*ANDN*、*ORN* 指令提供。*LDN* 指令将 *bit* 值取反，然后将结果装载于 CR 寄存器中并作为新的 CR 值；*ANDN* 指令将 *bit* 值取反后的结果和当前 CR 值进行“与”运算，并将运算结果作为新的 CR 值；*ORN* 指令将 *bit* 值取反后的结果和当前 CR 值进行“或”运算，并将运算结果作为新的 CR 值。

• 指令使用举例


LD
IL
%I0.0
%Q0.0
LDN %I0.0
ST %Q0.0
%I0.0
%I0.1
%Q0.1
LD %I0.0
ANDN %I0.1
ST %Q0.1
%I0.0
%I0.1
%Q0.2
LD %I0.0
ORN %I0.1
ST %Q0.2
时序图
I0.0
I0.1
Q0.0
Q0.1
Q0.2

6.2.3 普通线圈、复位线圈、置位线圈

- 指令及其操作数说明

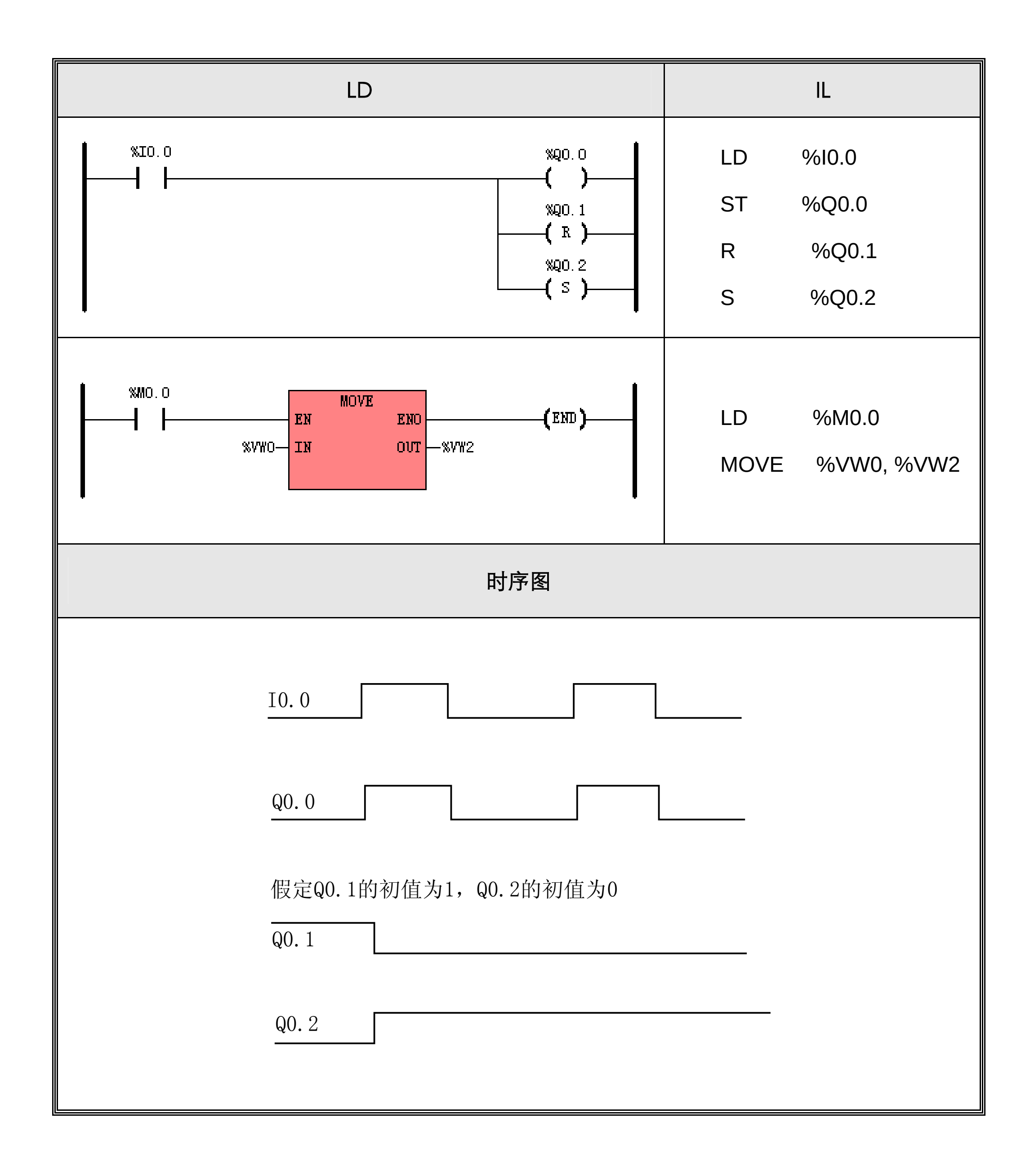
<table>
<tr><th></th><th>名 称</th><th>指令格式</th><th>影响 CR 值</th><th></th></tr>
<tr><td rowspan="4">LD</td><td>普通线圈</td><td>bit
—( )—</td><td rowspan="4"></td><td rowspan="7">☑ CPU304
☑ CPU306
☑ CPU308</td></tr>
<tr><td>复位线圈</td><td>bit
—(R)—</td></tr>
<tr><td>置位线圈</td><td>bit
—(S)—</td></tr>
<tr><td>结束线圈</td><td>—(END)—</td></tr>
<tr><td rowspan="3">IL</td><td>赋值</td><td>ST bit</td><td rowspan="3">U</td></tr>
<tr><td>复位</td><td>R bit</td></tr>
<tr><td>置位</td><td>S bit</td></tr>
</table>

操作数	输入/输出	数据类型	允许的内存区
bit	输出	BOOL	Q、V、M、SM、L

在 LD 中，普通线圈的作用是将线圈左侧能量流的值赋给 *bit* 。复位线圈的作用是：若线圈左侧能量流的值为 1，则 *bit* 值被置为 0，否则 *bit* 值保持不变。置位线圈的作用是：若线圈左侧能量流的值为 1，则 *bit* 值被置为 1，否则 *bit* 值保持不变。结束线圈的作用是：用于指示一个网络的结束，该指令仅为方便用户的编程而提供，并不执行具体的操作。

在 IL 中，*ST* 指令将当前的 CR 值赋给 *bit* 。 *R* 指令的作用是：若当前的 CR 值为 1，则 *bit* 值被置为 0，否则 *bit* 值保持不变。*S* 指令的作用是：若当前的 CR 值为 1，则 *bit* 值被置为 1，否则 *bit* 值保持不变。

• 指令使用举例


LD
IL
%I0.0
%Q0.0
%Q0.1
R
%Q0.2
S
LD %I0.0
ST %Q0.0
R %Q0.1
S %Q0.2
%M0.0
MOVE
EN
ENO
%VW0
IN
OUT
%VW2
END
LD %M0.0
MOVE %VW0, %VW2
时序图
I0.0
Q0.0
假定Q0.1的初值为1，Q0.2的初值为0
Q0.1
Q0.2

6.2.4　边沿（上升沿、下降沿）检测

- 指令及其参数说明

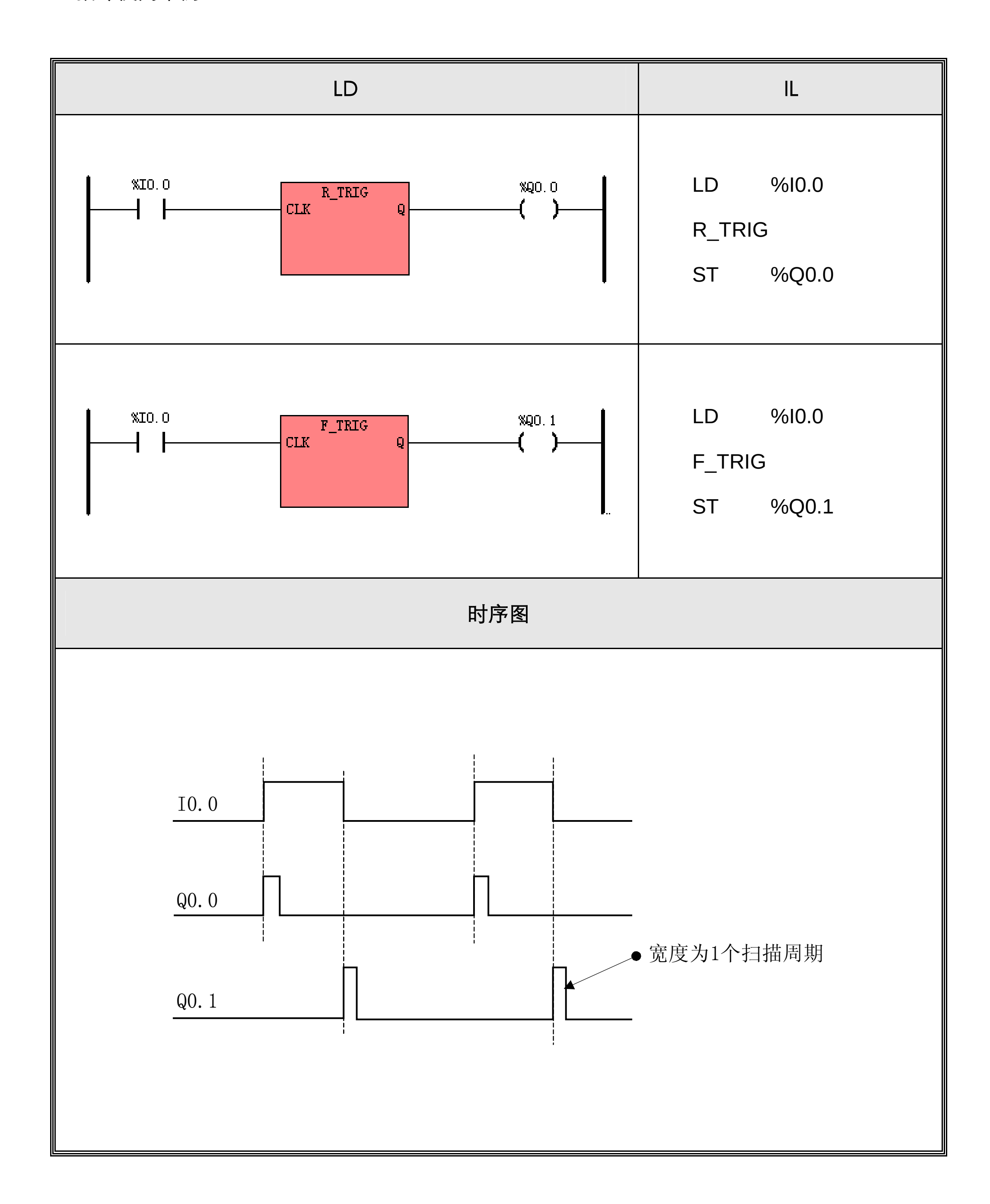
<table>
<tr><th></th><th>名 称</th><th>指 令</th><th>影响 CR 值</th><td rowspan="5">☑ CPU304
☑ CPU306
☑ CPU308</td></tr>
<tr><td rowspan="2">LD</td><td>上升沿检测</td><td>R_TRIG
CLK Q</td><td rowspan="2"></td></tr>
<tr><td>下降沿检测</td><td>F_TRIG
CLK Q</td></tr>
<tr><td rowspan="2">IL</td><td>上升沿检测</td><td>R_TRIG</td><td rowspan="2">P</td></tr>
<tr><td>下降沿检测</td><td>F_TRIG</td></tr>
</table>

参数	输入/输出	数据类型	允许的内存区
CLK（LD）	输入	BOOL	能量流
Q（LD）	输出	BOOL	能量流

在 LD 中，*R_TRIG* 用于检测 *CLK* 值的上升沿跳变：如果在 CPU 本次扫描过程中，*CLK* 值为 1，而在上一次扫描过程中，*CLK* 值为 0，则 *R_TRIG* 检测到了上升沿，将立即在 *Q* 输出 1，否则将在 *Q* 输出 0。 *F_TRIG* 用于检测 *CLK* 值的下降沿跳变，若检测到下降沿，将立即在 *Q* 输出 1，否则输出 0。

在 IL 中，*R_TRIG* 用于检测当前点 CR 值的上升沿跳变：如果在 CPU 本次扫描过程中，当前点的 CR 值为 1，而在上一次扫描过程中，当前点的 CR 值为 0，则 *R_TRIG* 检测到了上升沿，立即将 CR 值设置为 1，否则将 CR 值设置为 0。 *F_TRIG* 用于检测当前点 CR 值的下降沿跳变，若检测到下降沿，将立即将 CR 值设置 1，否则将 CR 值设置为 0。

• 指令使用举例


LD
IL
%I0.0
R_TRIG
CLK
Q
%Q0.0
LD %I0.0
R_TRIG
ST %Q0.0
%I0.0
F_TRIG
CLK
Q
%Q0.1
LD %I0.0
F_TRIG
ST %Q0.1
时序图
I0.0
Q0.0
Q0.1
宽度为1个扫描周期

6.2.5 括号修饰符

• 指令及其参数说明

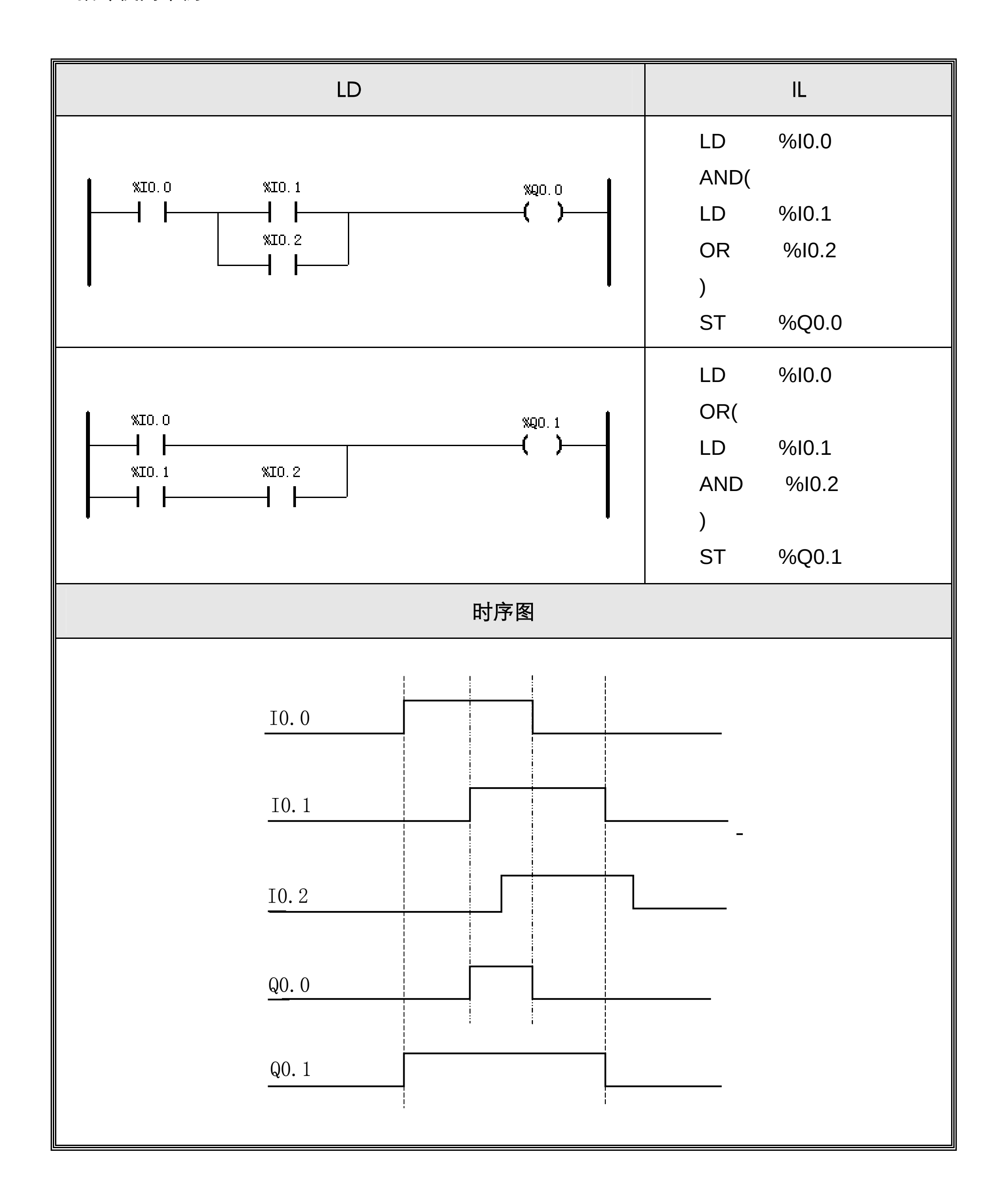
	名 称	指令格式	影响 CR 值	
IL		AND(	U	☑ CPU304 ☑ CPU306 ☑ CPU308
		OR(		
	结束括号	)	P	

括号修饰符只有在 IL 中提供。在 IL 语言中只有简单的表达式，而不可能象在 LD、ST 等语言中那样采用复杂的表达式作为操作数，因此 IEC61131-3 标准中定义了括号来处理一些复杂的表达式。“AND(”或者“OR(”都是和“)”配对使用的。

在 IL 程序中，执行“AND(”和“)”之间的指令之前，首先将当前 CR 值暂存，再执行括号中的指令，执行完之后将结果与刚才暂存的 CR 值进行与运算，并将运算结果作为新的 CR 值。

类似地，执行“OR(”和“)”之间的指令之前，首先将当前 CR 值暂存，再执行括号中的指令，执行完之后将结果与刚才暂存的 CR 值进行或运算，并将运算结果作为新的 CR 值。

- 指令使用举例


LD
IL
%I0.0
%I0.1
%I0.2
%Q0.0
LD %I0.0
AND(
LD %I0.1
OR %I0.2
)
ST %Q0.0
%I0.0
%Q0.1
%I0.1
%I0.2
LD %I0.0
OR(
LD %I0.1
AND %I0.2
)
ST %Q0.1
时序图
I0.0
I0.1
I0.2
Q0.0
Q0.1

6.3 赋值指令

6.3.1　MOVE（赋值）

• 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ CPU304 ☑ CPU306 ☑ CPU308
LD	赋值	MOVE EN　ENO IN　OUT		
IL	赋值	MOVE　IN, OUT	U	

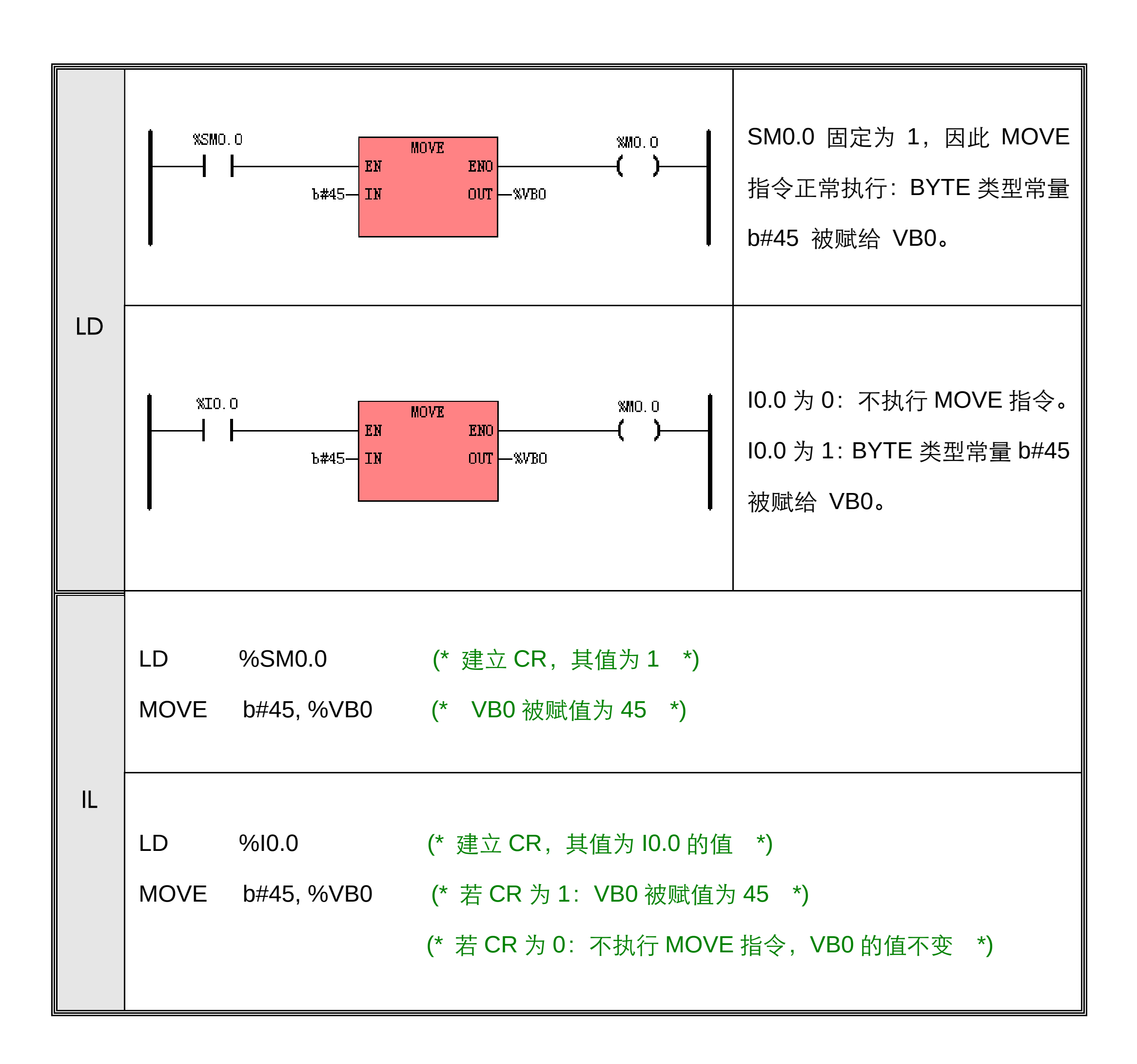
参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量
OUT	输出	BYTE、WORD、DWORD、INT、DINT、REAL	Q、M、V、L、SM、AQ

参数 *IN* 与 *OUT* 的数据类型必须一致。

在 LD 中，*EN* 作为使能端决定了是否进行赋值。如果 *EN* 为 1，则将输入 *IN* 的值赋给输出 *OUT*。

在 IL 中，由当前的 CR 值决定了是否进行赋值。如果当前 CR 值为 1，则将输入 *IN* 的值赋给输出 *OUT*。*MOVE* 指令的执行对 CR 值无影响。

• 指令使用举例



LD	%SM0.0 MOVE EN ENO b#45 IN OUT %VB0 %M0.0	SM0.0 固定为 1，因此 MOVE 指令正常执行：BYTE 类型常量 b#45 被赋给 VB0。
	%I0.0 MOVE EN ENO b#45 IN OUT %VB0 %M0.0	I0.0 为 0：不执行 MOVE 指令。I0.0 为 1：BYTE 类型常量 b#45 被赋给 VB0。
IL	LD %SM0.0 (* 建立 CR，其值为 1 *) MOVE b#45, %VB0 (* VB0 被赋值为 45 *)	
	LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *) MOVE b#45, %VB0 (* 若 CR 为 1：VB0 被赋值为 45 *) (* 若 CR 为 0：不执行 MOVE 指令，VB0 的值不变 *)	

6.3.2　BLKMOVE（块转移）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ CPU304 ☑ CPU306 ☑ CPU308
LD	块转移	BLKMOVE EN　ENO IN　OUT N		
IL	块转移	BLKMOVE　IN, OUT,N	U	

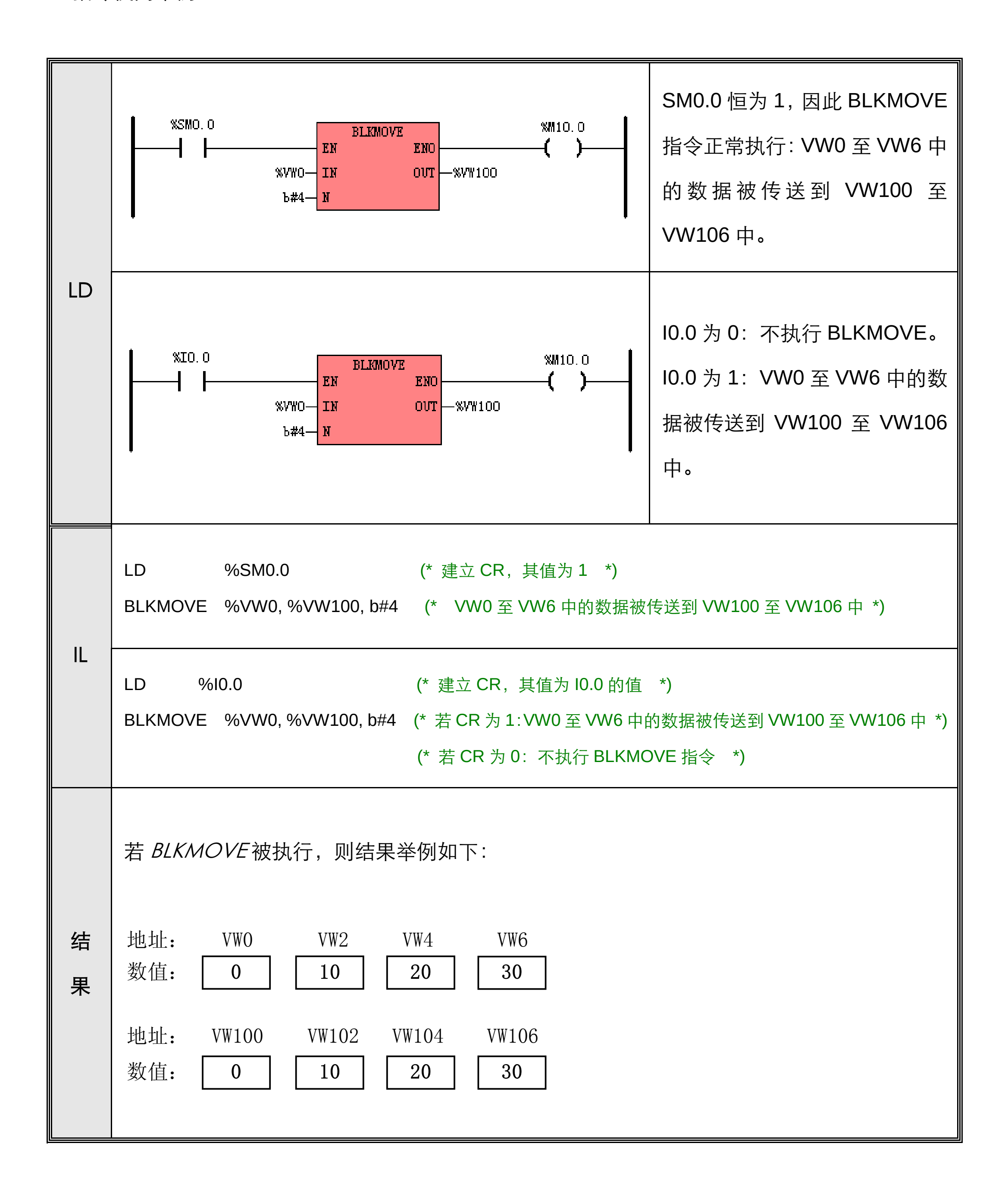
参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC
N	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD、INT、DINT、REAL	Q、M、V、L、SM、AQ

参数 *IN*、*N* 与 *OUT* 的数据类型必须一致。

在 LD 中，*EN* 作为使能端决定了是否执行 *BLKMOVE* 指令。如果 *EN* 为 1，则把从 *IN* 指定的地址开始的连续 *N* 个数据传送到从 *OUT* 指定的地址开始的连续 *N* 个地址空间中。

在 IL 中，由当前的 CR 值决定了是否执行 *BLKMOVE* 指令。如果当前 CR 值为 1，则把从 *IN* 指定的地址开始的连 *N* 个数据传送到从 *OUT* 指定的地址开始的连续 *N* 个地址空间中。*BLKMOVE* 指令的执行对 CR 值无影响。

• 指令使用举例



LD	%SM0.0 — BLKMOVE (EN, ENO; %VW0 — IN, OUT — %VW100; b#4 — N) — %M10.0 ( )	SM0.0 恒为 1，因此 BLKMOVE 指令正常执行：VW0 至 VW6 中的数据被传送到 VW100 至 VW106 中。
	%I0.0 — BLKMOVE (EN, ENO; %VW0 — IN, OUT — %VW100; b#4 — N) — %M10.0 ( )	I0.0 为 0：不执行 BLKMOVE。I0.0 为 1：VW0 至 VW6 中的数据被传送到 VW100 至 VW106 中。
IL	LD　%SM0.0　(* 建立 CR，其值为 1 *) BLKMOVE　%VW0, %VW100, b#4　(* VW0 至 VW6 中的数据被传送到 VW100 至 VW106 中 *)	
	LD　%I0.0　(* 建立 CR，其值为 I0.0 的值 *) BLKMOVE　%VW0, %VW100, b#4　(* 若 CR 为 1:VW0 至 VW6 中的数据被传送到 VW100 至 VW106 中 *) (* 若 CR 为 0：不执行 BLKMOVE 指令 *)	
结果	若 *BLKMOVE* 被执行，则结果举例如下：	

地址：	VW0	VW2	VW4	VW6
数值：	0	10	20	30

地址：	VW100	VW102	VW104	VW106
数值：	0	10	20	30

6.4 比较指令

注意：对于所有的比较指令，BYTE 类型的比较是无符号比较，其余的比较均为有符号比较。

6.4.1 GT（大于）

• 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	大于	GT EN OUT IN1 IN2		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	大于	GT IN1, IN2	P	

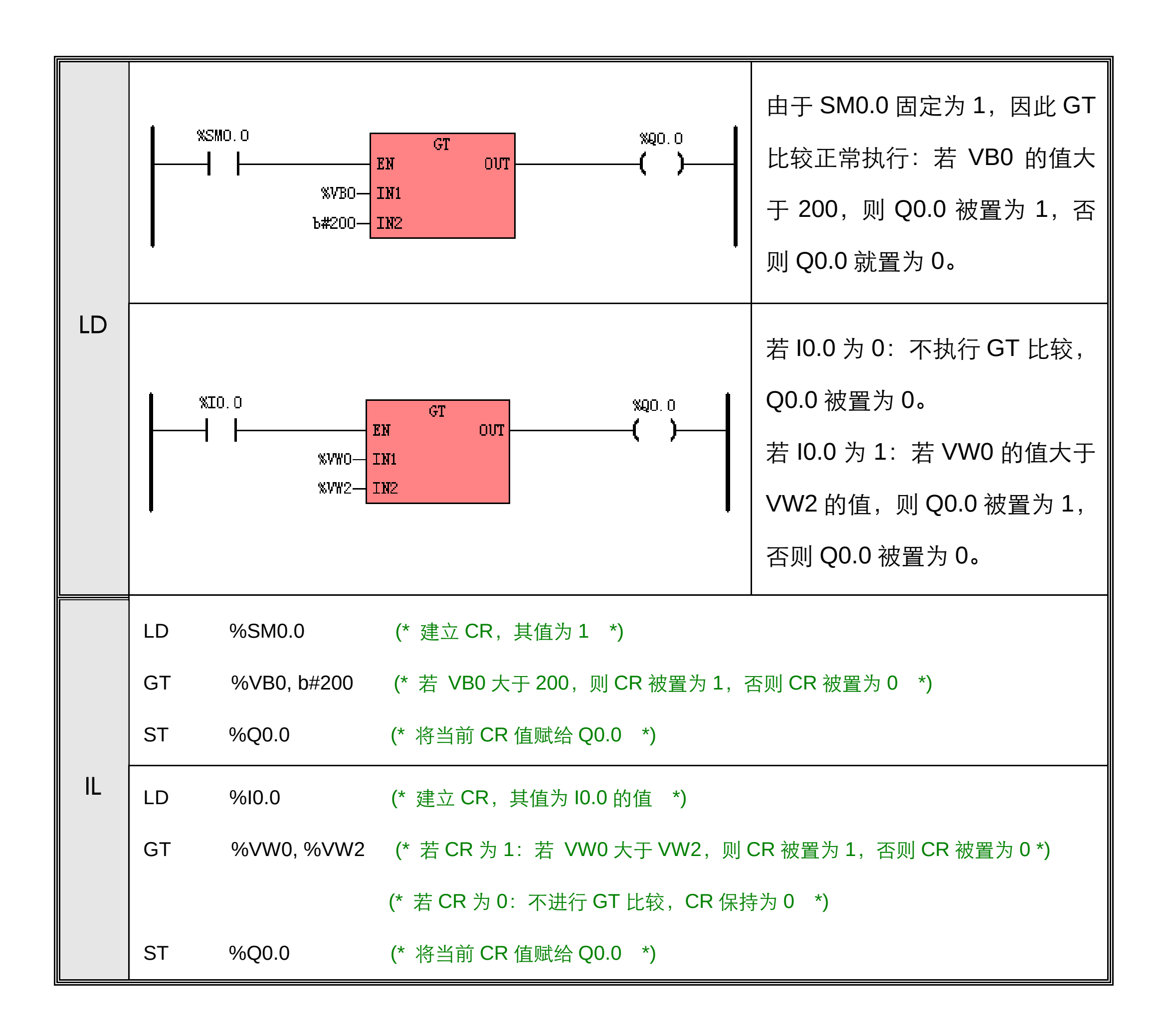
参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量
IN2	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量
OUT（LD）	输出	BOOL	能量流

参数 IN1、IN2 的数据类型必须一致。

在 LD 中， *EN* 作为使能端决定了是否进行 *GT* 比较。如果 *EN* 为 1，那么若 *IN1* 大于 *IN2*，则在 *OUT* 输出为 1，否则输出为 0；若 *EN* 为 0，则不进行 *GT* 比较，在 *OUT* 端必然输出为 0。

在 IL 中，由当前的 CR 值决定了是否进行 *GT* 比较。如果当前 CR 值为 1，那么若 *IN1* 大于 *IN2*，则将 CR 值设置为 1，否则将 CR 值设置为 0；如果当前 CR 值为 0，则不进行比较。

• 指令使用举例



LD	%SM0.0 — GT (EN, IN1: %VB0, IN2: b#200, OUT) — %Q0.0	由于 SM0.0 固定为 1，因此 GT 比较正常执行：若 VB0 的值大于 200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。
	%I0.0 — GT (EN, IN1: %VW0, IN2: %VW2, OUT) — %Q0.0	若 I0.0 为 0：不执行 GT 比较，Q0.0 被置为 0。 若 I0.0 为 1：若 VW0 的值大于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。
IL	LD %SM0.0 (* 建立 CR，其值为 1 *) GT %VB0, b#200 (* 若 VB0 大于 200，则 CR 被置为 1，否则 CR 被置为 0 *) ST %Q0.0 (* 将当前 CR 值赋给 Q0.0 *)	
	LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *) GT %VW0, %VW2 (* 若 CR 为 1：若 VW0 大于 VW2，则 CR 被置为 1，否则 CR 被置为 0 *) (* 若 CR 为 0：不进行 GT 比较，CR 保持为 0 *) ST %Q0.0 (* 将当前 CR 值赋给 Q0.0 *)	

6.4.2　GE（大于等于）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ CPU304 ☑ CPU306 ☑ CPU308
LD	大于等于	GE EN OUT IN1 IN2		
IL	大于等于	GE　IN1, IN2	P	

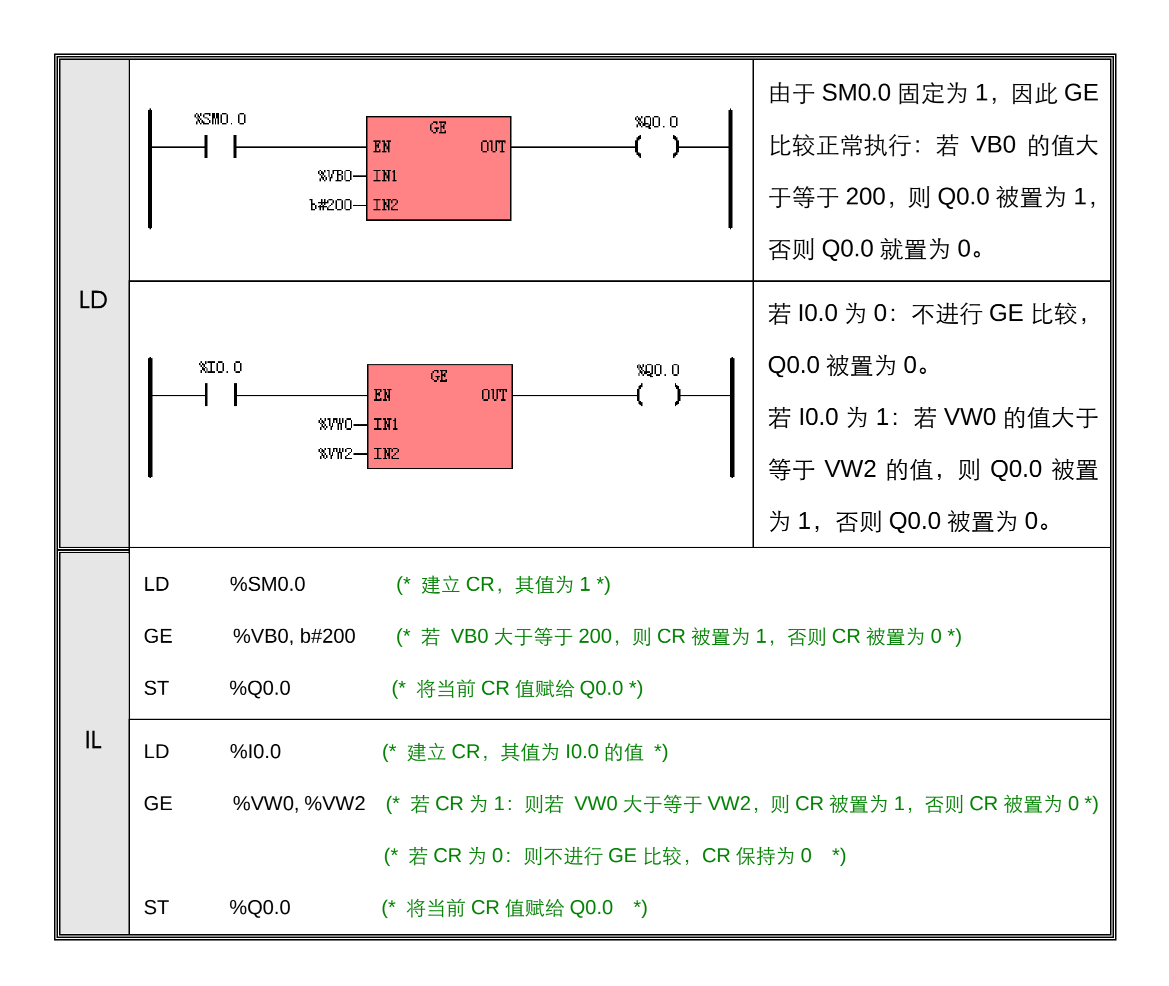
参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量
IN2	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量
OUT（LD）	输出	BOOL	能量流

参数 IN1、IN2 的数据类型必须一致。

在 LD 中，*EN* 作为使能端决定了 *GE* 比较是否被执行。如果 *EN* 为 1，那么若 *IN1* 大于等于 *IN2*，则在 *OUT* 输出为 1，否则输出为 0；若 *EN* 为 0，则不进行 *GE* 比较，在 *OUT* 端必然输出为 0。

在 IL 中，由当前的 CR 值决定了是否进行 *GE* 比较。如果当前 CR 值为 1，那么若 *IN1* 大于等于 *IN2*，则将 CR 值设置为 1，否则将 CR 值设置为 0；如果当前 CR 值为 0，则不进行比较，CR 值也保持不变。

• 指令使用举例



LD	%SM0.0 —┤ ├— GE (EN, OUT; %VB0—IN1, b#200—IN2) —( )— %Q0.0	由于 SM0.0 固定为 1，因此 GE 比较正常执行：若 VB0 的值大于等于 200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。
LD	%I0.0 —┤ ├— GE (EN, OUT; %VW0—IN1, %VW2—IN2) —( )— %Q0.0	若 I0.0 为 0：不进行 GE 比较，Q0.0 被置为 0。 若 I0.0 为 1：若 VW0 的值大于等于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。
IL	LD　%SM0.0　(* 建立 CR，其值为 1 *) GE　%VB0, b#200　(* 若 VB0 大于等于 200，则 CR 被置为 1，否则 CR 被置为 0 *) ST　%Q0.0　(* 将当前 CR 值赋给 Q0.0 *)	
IL	LD　%I0.0　(* 建立 CR，其值为 I0.0 的值 *) GE　%VW0, %VW2　(* 若 CR 为 1：则若 VW0 大于等于 VW2，则 CR 被置为 1，否则 CR 被置为 0 *) (* 若 CR 为 0：则不进行 GE 比较，CR 保持为 0 *) ST　%Q0.0　(* 将当前 CR 值赋给 Q0.0 *)	

6.4.3 EQ（等于）

• 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ CPU304 ☑ CPU306 ☑ CPU308
LD	等于	EQ: EN, IN1, IN2, OUT		
IL	等于	EQ IN1, IN2	P	

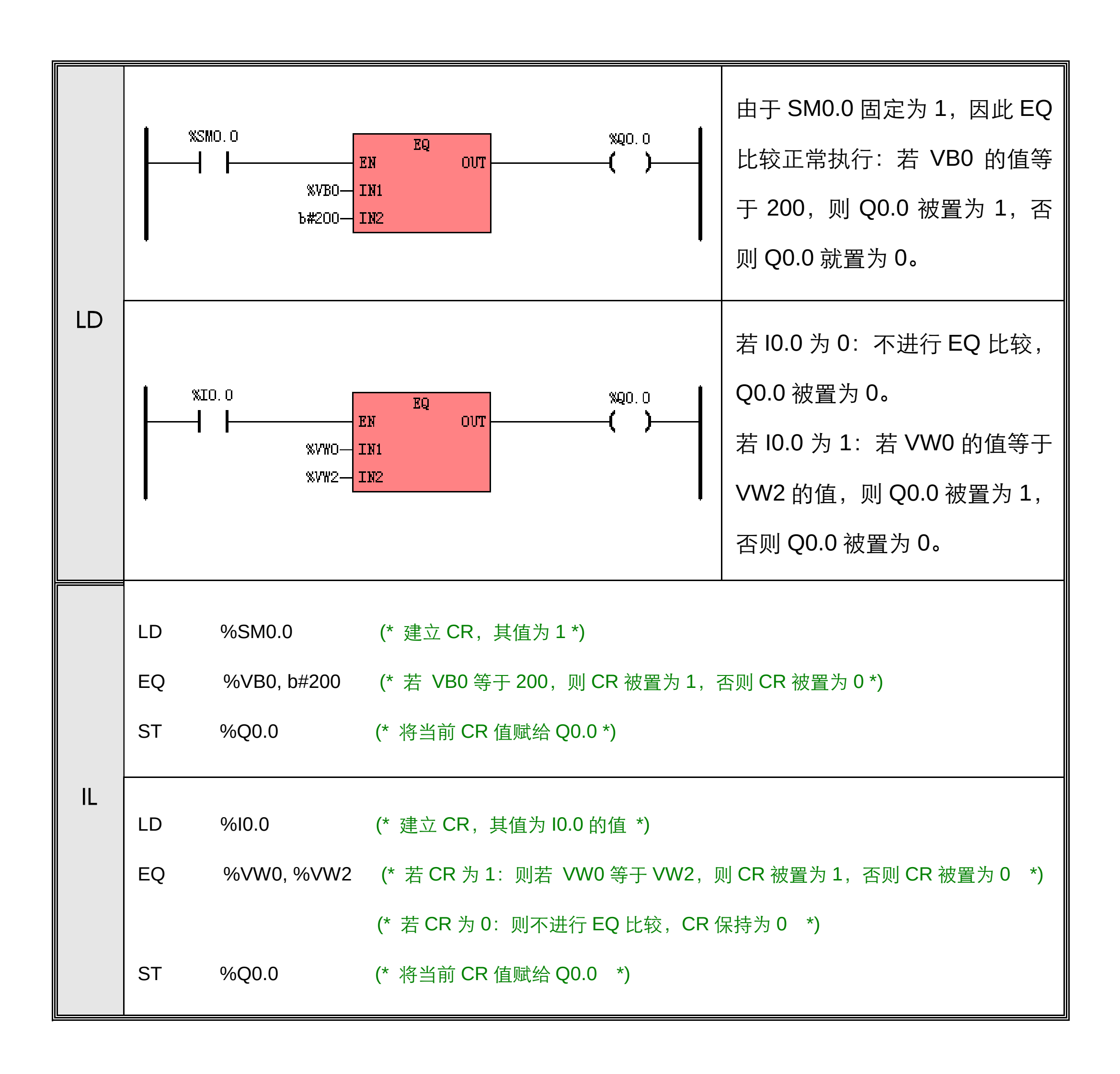
参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量
IN2	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量
OUT（LD）	输出	BOOL	能量流

参数 IN1、IN2 的数据类型必须一致。

在 LD 中， *EN* 作为使能端决定了 *EQ* 比较是否被执行。如果 *EN* 为 1，那么若 *IN1* 等于 *IN2*，则在 *OUT* 输出为 1，否则输出为 0；若 *EN* 为 0，则不进行 *EQ* 比较，在 *OUT* 端必然输出为 0。

在 IL 中，由 CR 值决定了是否进行 *EQ* 比较。如果 CR 值为 1，那么若 *IN1* 等于 *IN2*，则将 CR 值设置为 1，否则将 CR 值设置为 0；如果 CR 值为 0，则不进行比较，CR 值也保持不变。

• 指令使用举例



	程序	说明
LD	（梯形图：%SM0.0 → EQ（IN1 %VB0，IN2 b#200）→ %Q0.0）	由于 SM0.0 固定为 1，因此 EQ 比较正常执行：若 VB0 的值等于 200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。
	（梯形图：%I0.0 → EQ（IN1 %VW0，IN2 %VW2）→ %Q0.0）	若 I0.0 为 0：不进行 EQ 比较，Q0.0 被置为 0。 若 I0.0 为 1：若 VW0 的值等于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。
IL	LD %SM0.0 (* 建立 CR，其值为 1 *) EQ %VB0, b#200 (* 若 VB0 等于 200，则 CR 被置为 1，否则 CR 被置为 0 *) ST %Q0.0 (* 将当前 CR 值赋给 Q0.0 *)	
	LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *) EQ %VW0, %VW2 (* 若 CR 为 1：则若 VW0 等于 VW2，则 CR 被置为 1，否则 CR 被置为 0 *) (* 若 CR 为 0：则不进行 EQ 比较，CR 保持为 0 *) ST %Q0.0 (* 将当前 CR 值赋给 Q0.0 *)	

6.4.4 NE（不等于）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	不等于	NE（EN、IN1、IN2、OUT）		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	不等于	NE IN1, IN2	P	

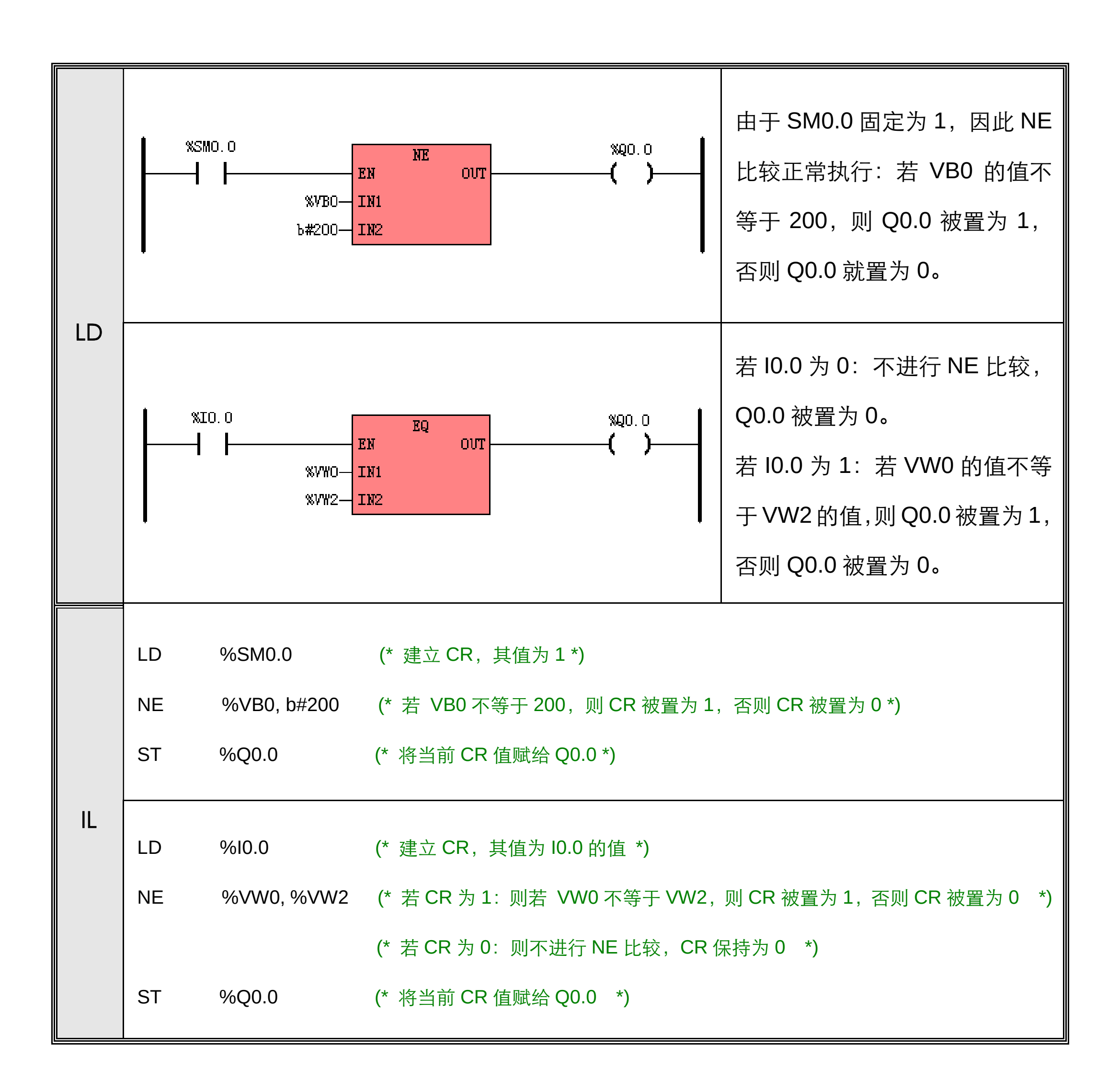
参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量
IN2	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量
OUT（LD）	输出	BOOL	能量流

参数 IN1、IN2 的数据类型必须一致。

在 LD 中，*EN* 作为使能端决定了 *NE* 比较是否被执行。如果 *EN* 为 1，那么若 *IN1* 不等于 *IN2*，则在 *OUT* 输出为 1，否则输出为 0；若 *EN* 为 0，则不进行 *NE* 比较，在 *OUT* 端必然输出为 0。

在 IL 中，由 CR 值决定了是否进行 *NE* 比较。如果 CR 值为 1，那么若 *IN1* 不等于 *IN2*，则将 CR 值设置为 1，否则将 CR 值设置为 0；如果 CR 值为 0，则不进行比较，CR 值也保持不变。

• 指令使用举例



	程序	说明
LD	%SM0.0 ─┤ ├─ NE (EN, IN1: %VB0, IN2: b#200, OUT) ─( )─ %Q0.0	由于 SM0.0 固定为 1，因此 NE 比较正常执行：若 VB0 的值不等于 200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。
	%I0.0 ─┤ ├─ EQ (EN, IN1: %VW0, IN2: %VW2, OUT) ─( )─ %Q0.0	若 I0.0 为 0：不进行 NE 比较，Q0.0 被置为 0。 若 I0.0 为 1：若 VW0 的值不等于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。
IL	LD　%SM0.0　(* 建立 CR，其值为 1 *) NE　%VB0, b#200　(* 若 VB0 不等于 200，则 CR 被置为 1，否则 CR 被置为 0 *) ST　%Q0.0　(* 将当前 CR 值赋给 Q0.0 *)	
	LD　%I0.0　(* 建立 CR，其值为 I0.0 的值 *) NE　%VW0, %VW2　(* 若 CR 为 1：则若 VW0 不等于 VW2，则 CR 被置为 1，否则 CR 被置为 0 *) (* 若 CR 为 0：则不进行 NE 比较，CR 保持为 0 *) ST　%Q0.0　(* 将当前 CR 值赋给 Q0.0 *)	

6.4.5 LT（小于）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	小于	LT: EN, OUT, IN1, IN2		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	小于	LT IN1, IN2	P	

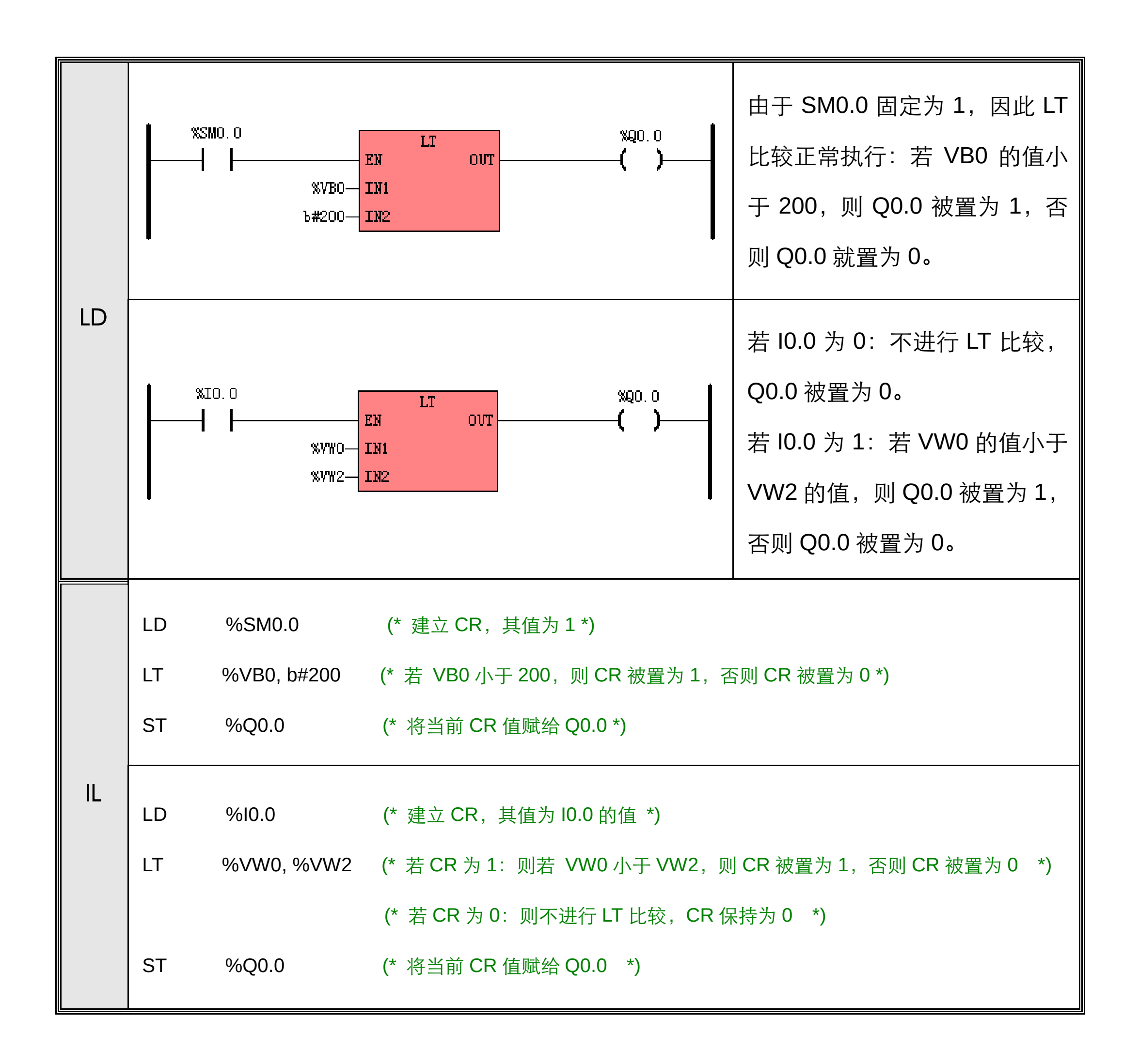
参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常
IN2	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常
OUT（LD）	输出	BOOL	能量流

参数 IN1、IN2 的数据类型必须一致。

在 LD 中， *EN* 作为使能端决定了 *LT* 比较是否被执行。如果 *EN* 为 1，那么若 *IN1* 小于 *IN2* ，则在 *OUT* 输出为 1，否则输出为 0；若 *EN* 为 0，则不进行 *LT* 比较，在 *OUT* 端必然输出为 0。

在 IL 中，由 CR 值决定了是否进行 *LT* 比较。如果 CR 值为 1，那么若 *IN1* 小于 *IN2* ，则将 CR 值设置为 1，否则将 CR 值设置为 0；如果 CR 值为 0，则不进行比较，CR 值也保持不变。

• 指令使用举例



LD	（梯形图）	由于 SM0.0 固定为 1，因此 LT 比较正常执行：若 VB0 的值小于 200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。
LD	（梯形图）	若 I0.0 为 0：不进行 LT 比较，Q0.0 被置为 0。 若 I0.0 为 1：若 VW0 的值小于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。
IL	LD　%SM0.0　(* 建立 CR，其值为 1 *) LT　%VB0, b#200　(* 若 VB0 小于 200，则 CR 被置为 1，否则 CR 被置为 0 *) ST　%Q0.0　(* 将当前 CR 值赋给 Q0.0 *)	
IL	LD　%I0.0　(* 建立 CR，其值为 I0.0 的值 *) LT　%VW0, %VW2　(* 若 CR 为 1：则若 VW0 小于 VW2，则 CR 被置为 1，否则 CR 被置为 0 *) (* 若 CR 为 0：则不进行 LT 比较，CR 保持为 0 *) ST　%Q0.0　(* 将当前 CR 值赋给 Q0.0 *)	

6.4.6 LE（小于等于）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ CPU304 ☑ CPU306 ☑ CPU308
LD	小于等于	LE EN OUT IN1 IN2		
IL	小于等于	LE IN1, IN2	P	

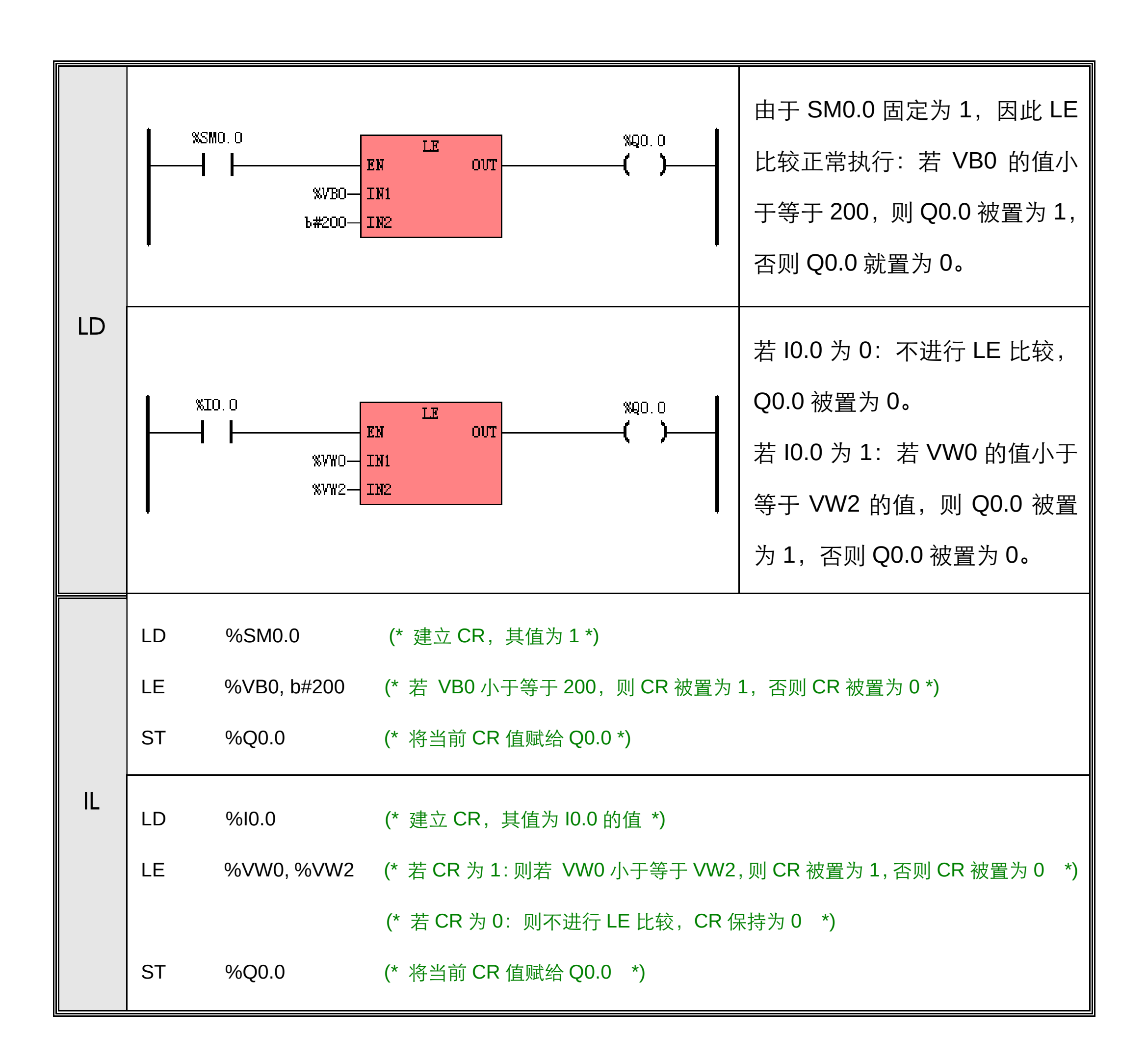
参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常
IN2	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常
OUT（LD）	输出	BOOL	能量流

参数 IN1、IN2 的数据类型必须一致。

在 LD 中， *EN* 作为使能端决定了 *LE* 比较是否被执行。如果 *EN* 为 1，那么若 *IN1* 小于等于 *IN2* ，则在 *OUT* 输出为 1，否则输出为 0；若 *EN* 为 0，则不进行 *LE* 比较，在 *OUT* 端必然输出为 0。

在 IL 中，由 CR 值决定了是否进行 *LE* 比较。如果当前 CR 值为 1，那么若 *IN1* 小于等于 *IN2* ，则将 CR 值设置为 1，否则将 CR 值设置为 0；如果 CR 值为 0，则不进行比较，CR 值也保持不变。

• 指令使用举例



	程序	说明
LD	%SM0.0 —| |— [LE: EN, OUT; %VB0—IN1; b#200—IN2] —(%Q0.0)	由于 SM0.0 固定为 1，因此 LE 比较正常执行：若 VB0 的值小于等于 200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。
	%I0.0 —| |— [LE: EN, OUT; %VW0—IN1; %VW2—IN2] —(%Q0.0)	若 I0.0 为 0：不进行 LE 比较，Q0.0 被置为 0。 若 I0.0 为 1：若 VW0 的值小于等于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。
IL	LD　%SM0.0　(* 建立 CR，其值为 1 *) LE　%VB0, b#200　(* 若 VB0 小于等于 200，则 CR 被置为 1，否则 CR 被置为 0 *) ST　%Q0.0　(* 将当前 CR 值赋给 Q0.0 *)	
	LD　%I0.0　(* 建立 CR，其值为 I0.0 的值 *) LE　%VW0, %VW2　(* 若 CR 为 1: 则若 VW0 小于等于 VW2, 则 CR 被置为 1, 否则 CR 被置为 0 *) (* 若 CR 为 0：则不进行 LE 比较，CR 保持为 0 *) ST　%Q0.0　(* 将当前 CR 值赋给 Q0.0 *)	

6.5 逻辑运算

所谓逻辑运算是指对字节型或者整数进行二进制位的运算。

6.5.1 NOT（按位取反）

• 指令及其操作数说明

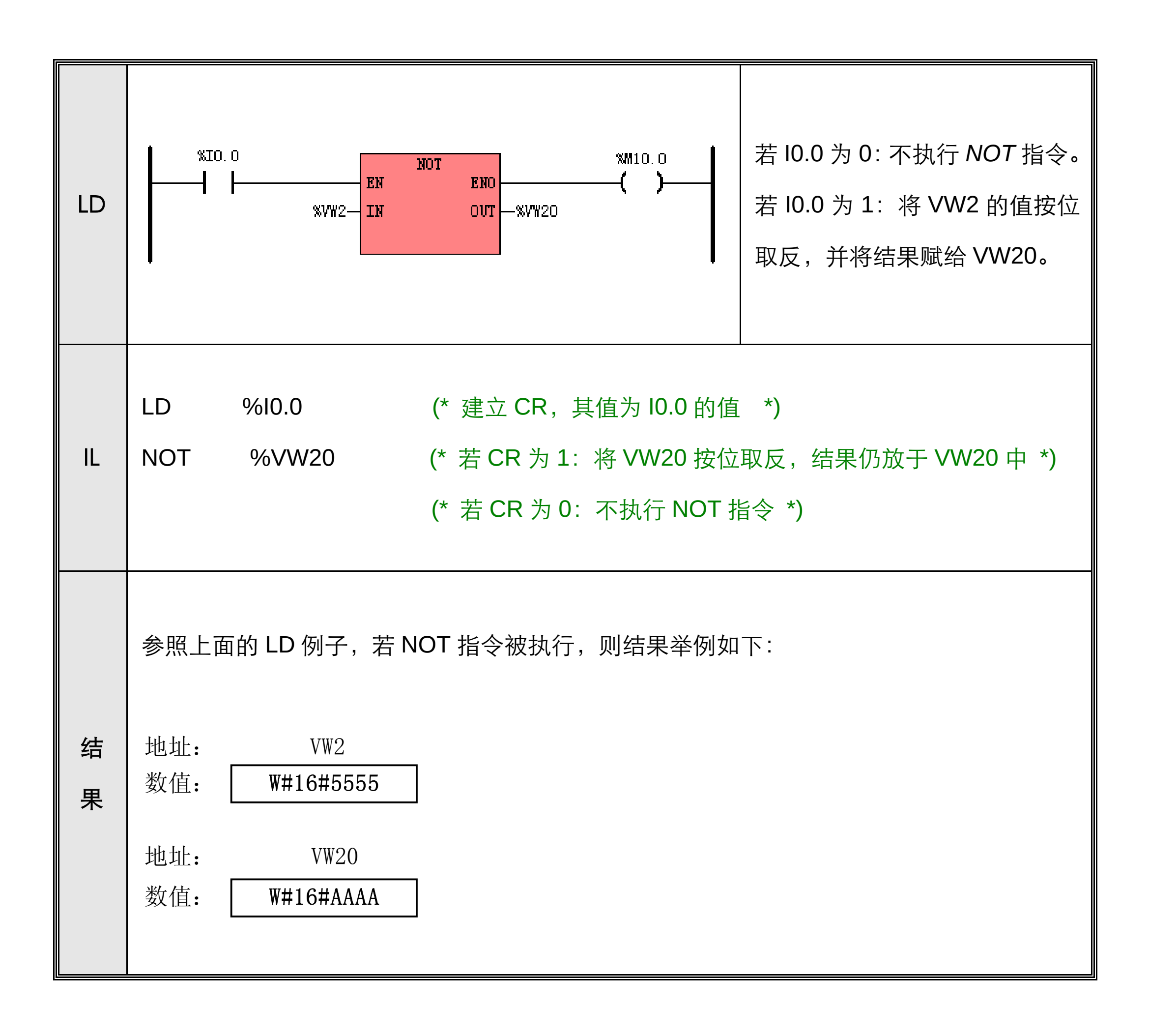
	名称	指令格式	影响 CR 值	☑ CPU304 ☑ CPU306 ☑ CPU308
LD	按位取反	NOT EN ENO IN OUT		
IL	按位取反	NOT OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM

在 LD 中， *EN* 作为使能端决定了是否执行 *NOT* 指令。如果 *EN* 为 1，那么将输入 *IN* 的每一个二进制位都取反，并将结果置于 *OUT* 中。参数 *IN*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *NOT* 指令。如果 CR 值为 1，那么将参数 *OUT* 的每一个二进制位都取反，结果仍旧置于 *OUT* 中。*NOT* 指令的执行不影响 CR 值。

• 指令使用举例



LD	%I0.0　NOT　EN ENO　%M10.0　%VW2 — IN OUT — %VW20	若 I0.0 为 0: 不执行 *NOT* 指令。 若 I0.0 为 1：将 VW2 的值按位取反，并将结果赋给 VW20。
IL	LD　%I0.0　(* 建立 CR，其值为 I0.0 的值 *) NOT　%VW20　(* 若 CR 为 1：将 VW20 按位取反，结果仍放于 VW20 中 *) (* 若 CR 为 0：不执行 NOT 指令 *)	
结果	参照上面的 LD 例子，若 NOT 指令被执行，则结果举例如下: 地址：VW2 数值：W#16#5555 地址：VW20 数值：W#16#AAAA	

6.5.2　AND（按位与）

- 指令及其操作数说明

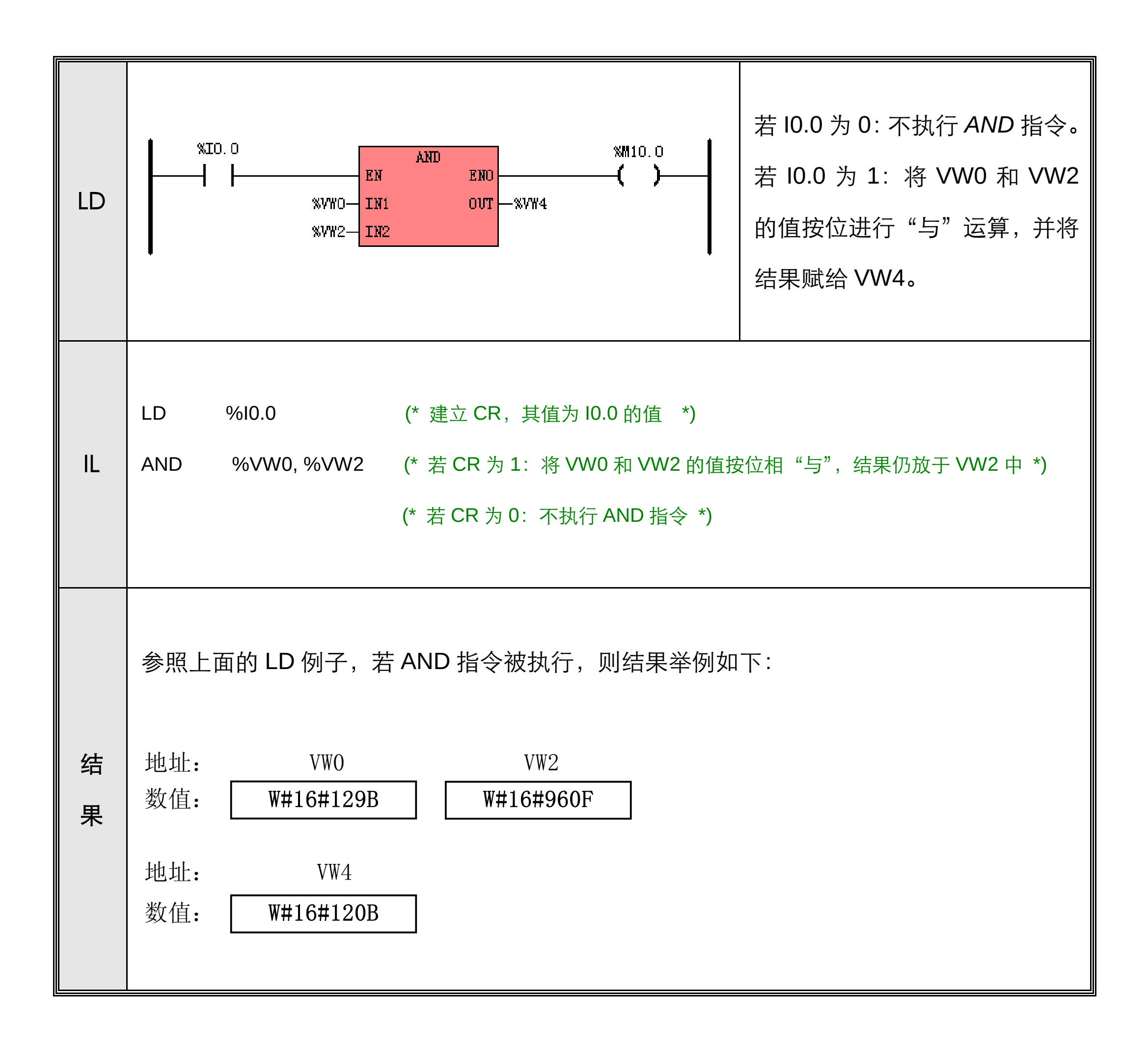
	名称	指令格式	影响 CR 值	☑ CPU304 ☑ CPU306 ☑ CPU308
LD	按位与	AND EN　ENO IN1　OUT IN2		
IL	按位与	AND　　IN, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM

在 LD 中，*EN* 作为使能端决定了是否执行 *AND* 指令。如果 *EN* 为 1，则将输入 *IN1 和 IN2* 按二进制位进行“与”运算后，将结果置于 *OUT* 中。参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *AND* 指令。如果 CR 为 1，那么将参数 *IN 和 OUT* 按二进制位进行“与”运算后，将结果仍旧置于 *OUT* 中。参数 *IN*、*OUT* 的数据类型必须一致。*AND* 指令的执行不影响 CR 值。

- 指令使用举例



LD	%I0.0 ─┤ ├─ [AND: EN, ENO; %VW0─IN1, OUT─%VW4; %VW2─IN2] ─( )─ %M10.0	若 I0.0 为 0: 不执行 *AND* 指令。 若 I0.0 为 1: 将 VW0 和 VW2 的值按位进行“与”运算，并将结果赋给 VW4。
IL	LD　%I0.0　(* 建立 CR，其值为 I0.0 的值 *) AND　%VW0, %VW2　(* 若 CR 为 1: 将 VW0 和 VW2 的值按位相“与”，结果仍放于 VW2 中 *) (* 若 CR 为 0: 不执行 AND 指令 *)	
结果	参照上面的 LD 例子，若 AND 指令被执行，则结果举例如下: 地址: VW0 / VW2 数值: W#16#129B / W#16#960F 地址: VW4 数值: W#16#120B	

6.5.3　ANDN（按位与非）

- 指令及其操作数说明

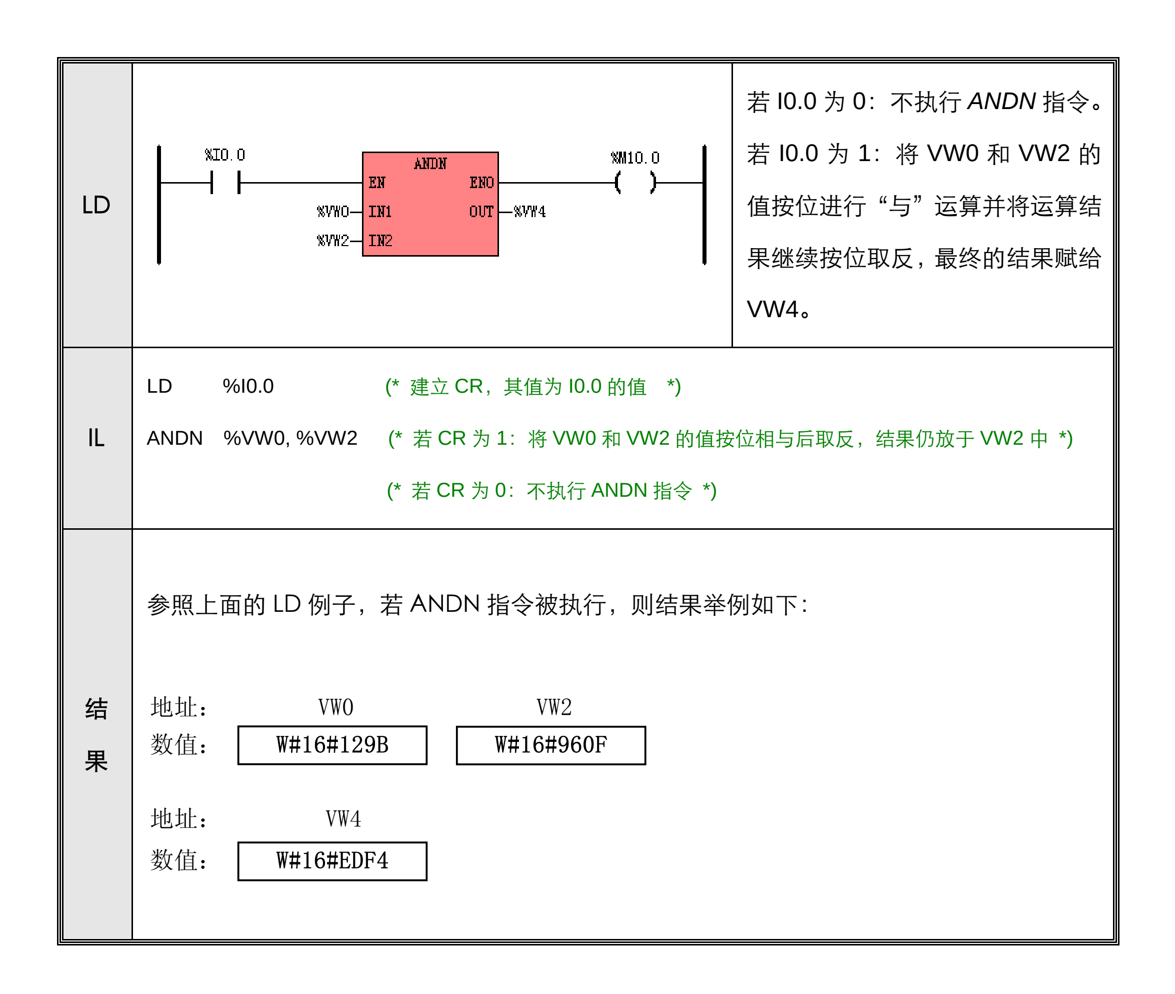
	名称	指令格式	影响 CR 值	
LD	按位与非	ANDN (EN, IN1, IN2, ENO, OUT)		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	按位与非	ANDN　　IN, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM

在 LD 中，*EN* 作为使能端决定了是否执行 *ANDN* 指令。如果 *EN* 为 1，那么将输入 *IN1 和 IN2* 按二进制位进行“与”运算并将所得结果继续按位取反，最终的结果被置于 *OUT* 中。参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *ANDN* 指令。如果 CR 为 1，那么那么将参数 *IN 和 OUT* 按二进制位进行“与”运算并将所得结果继续按位取反，最终的结果被置于 *OUT* 中。参数 *IN*、*OUT* 的数据类型必须一致。*ANDN* 指令的执行不影响 CR 值。

• 指令使用举例



LD	%I0.0 / ANDN / EN / ENO / %VW0—IN1 / OUT—%VW4 / %VW2—IN2 / %M10.0	若 I0.0 为 0：不执行 *ANDN* 指令。 若 I0.0 为 1：将 VW0 和 VW2 的值按位进行“与”运算并将运算结果继续按位取反，最终的结果赋给 VW4。
IL	LD　　%I0.0　　(* 建立 CR，其值为 I0.0 的值 *) ANDN　%VW0, %VW2　(* 若 CR 为 1：将 VW0 和 VW2 的值按位相与后取反，结果仍放于 VW2 中 *) (* 若 CR 为 0：不执行 ANDN 指令 *)	
结果	参照上面的 LD 例子，若 ANDN 指令被执行，则结果举例如下： 地址：VW0　VW2 数值：W#16#129B　W#16#960F 地址：VW4 数值：W#16#EDF4	

6.5.4 OR（按位或）

• 指令及其操作数说明

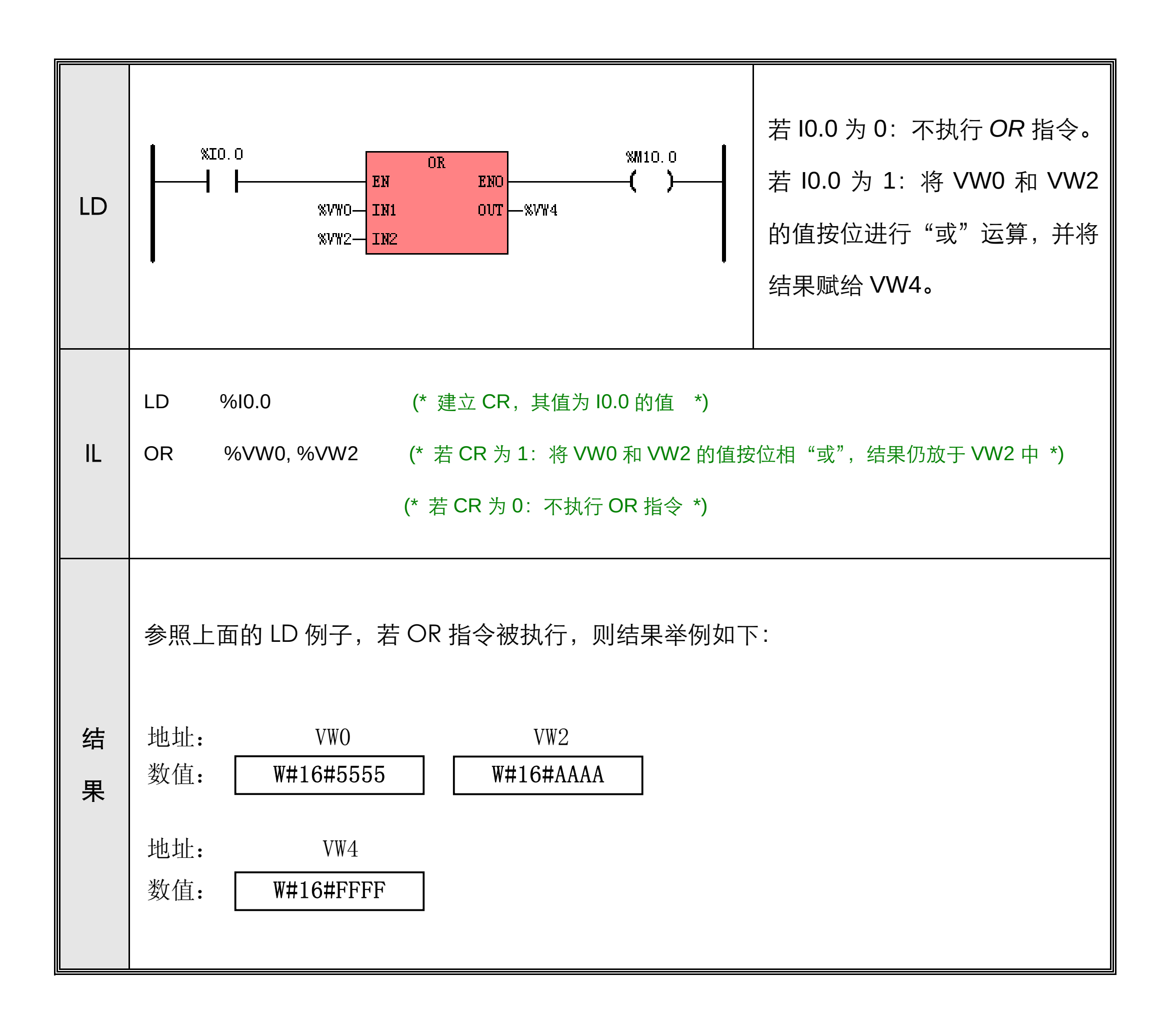
<table>
<tr><th></th><th>名称</th><th>指令格式</th><th>影响 CR 值</th><td rowspan="3">☑ CPU304
☑ CPU306
☑ CPU308</td></tr>
<tr><td>LD</td><td>按位或</td><td>OR
EN ENO
IN1 OUT
IN2</td><td></td></tr>
<tr><td>IL</td><td>按位或</td><td>OR IN, OUT</td><td>U</td></tr>
</table>

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM

在 LD 中，*EN* 作为使能端决定了是否执行 *OR* 指令。如果 *EN* 为 1，则将输入 *IN1 和 IN2* 按二进制位进行“或”运算后，将结果置于 *OUT* 中。参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *OR* 指令。如果 CR 为 1，那么将参数 *IN 和 OUT* 按二进制位进行“或”运算后，将结果仍旧置于 *OUT* 中。参数 *IN*、*OUT* 的数据类型必须一致。*OR* 指令的执行不影响 CR 值。

• 指令使用举例


LD
%I0.0
OR
EN
ENO
%VW0
IN1
OUT
%VW4
%VW2
IN2
%M10.0
若 I0.0 为 0：不执行 OR 指令。
若 I0.0 为 1：将 VW0 和 VW2 的值按位进行“或”运算，并将结果赋给 VW4。
IL
LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
OR %VW0, %VW2 (* 若 CR 为 1：将 VW0 和 VW2 的值按相“或”，结果仍放于 VW2 中 *)
(* 若 CR 为 0：不执行 OR 指令 *)
结果
参照上面的 LD 例子，若 OR 指令被执行，则结果举例如下：
地址： VW0 VW2
数值： W#16#5555 W#16#AAAA
地址： VW4
数值： W#16#FFFF

6.5.5　ORN（按位或非）

- 指令及其操作数说明

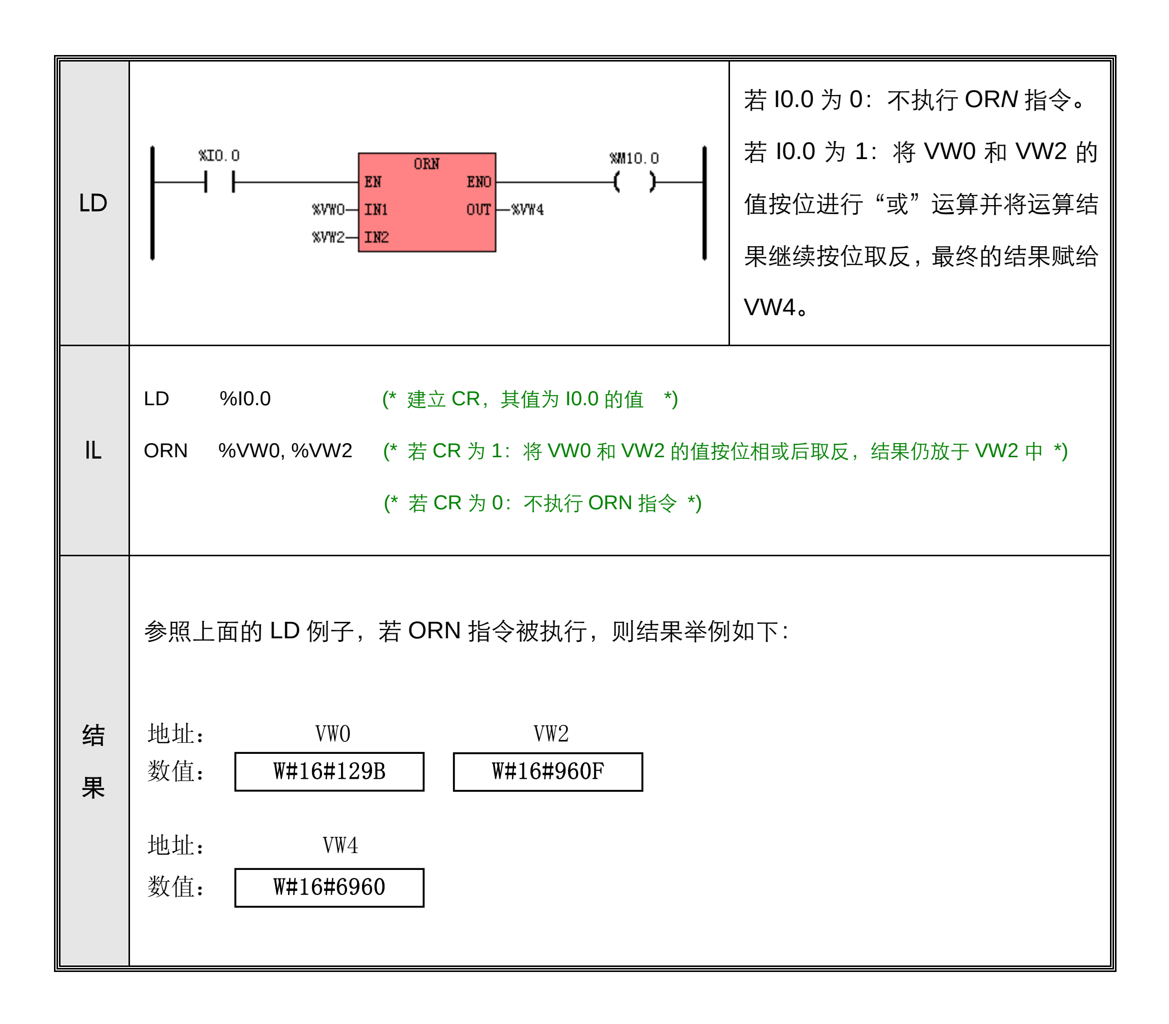
	名称	指令格式	影响 CR 值	☑ CPU304 ☑ CPU306 ☑ CPU308
LD	按位或非	ORN EN　ENO IN1　OUT IN2		
IL	按位或非	ORN　　IN, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM

在 LD 中，*EN* 作为使能端决定了是否执行 *ORN* 指令。如果 *EN* 为 1，那么将输入 *IN1 和 IN2* 按二进制位进行“或”运算并将所得结果继续按位取反，最终的结果被置于 *OUT* 中。参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *ORN* 指令。如果 CR 为 1，那么那么将参数 *IN 和 OUT* 按二进制位进行“或”运算并将所得结果继续按位取反，最终的结果被置于 *OUT* 中。参数 *IN*、*OUT* 的数据类型必须一致。*ANDN* 指令的执行不影响 CR 值。

• 指令使用举例



LD	%I0.0 ORN EN ENO %VW0 IN1 OUT %VW4 %VW2 IN2 %M10.0	若 I0.0 为 0：不执行 ORN 指令。 若 I0.0 为 1：将 VW0 和 VW2 的值按位进行“或”运算并将运算结果继续按位取反，最终的结果赋给 VW4。
IL	LD　　%I0.0　　(* 建立 CR，其值为 I0.0 的值　*) ORN　%VW0, %VW2　(* 若 CR 为 1：将 VW0 和 VW2 的值按位相或后取反，结果仍放于 VW2 中 *) (* 若 CR 为 0：不执行 ORN 指令 *)	
结果	参照上面的 LD 例子，若 ORN 指令被执行，则结果举例如下： 地址：VW0　VW2 数值：W#16#129B　W#16#960F 地址：VW4 数值：W#16#6960	

6.5.6　XOR（按位异或）

- 指令及其操作数说明

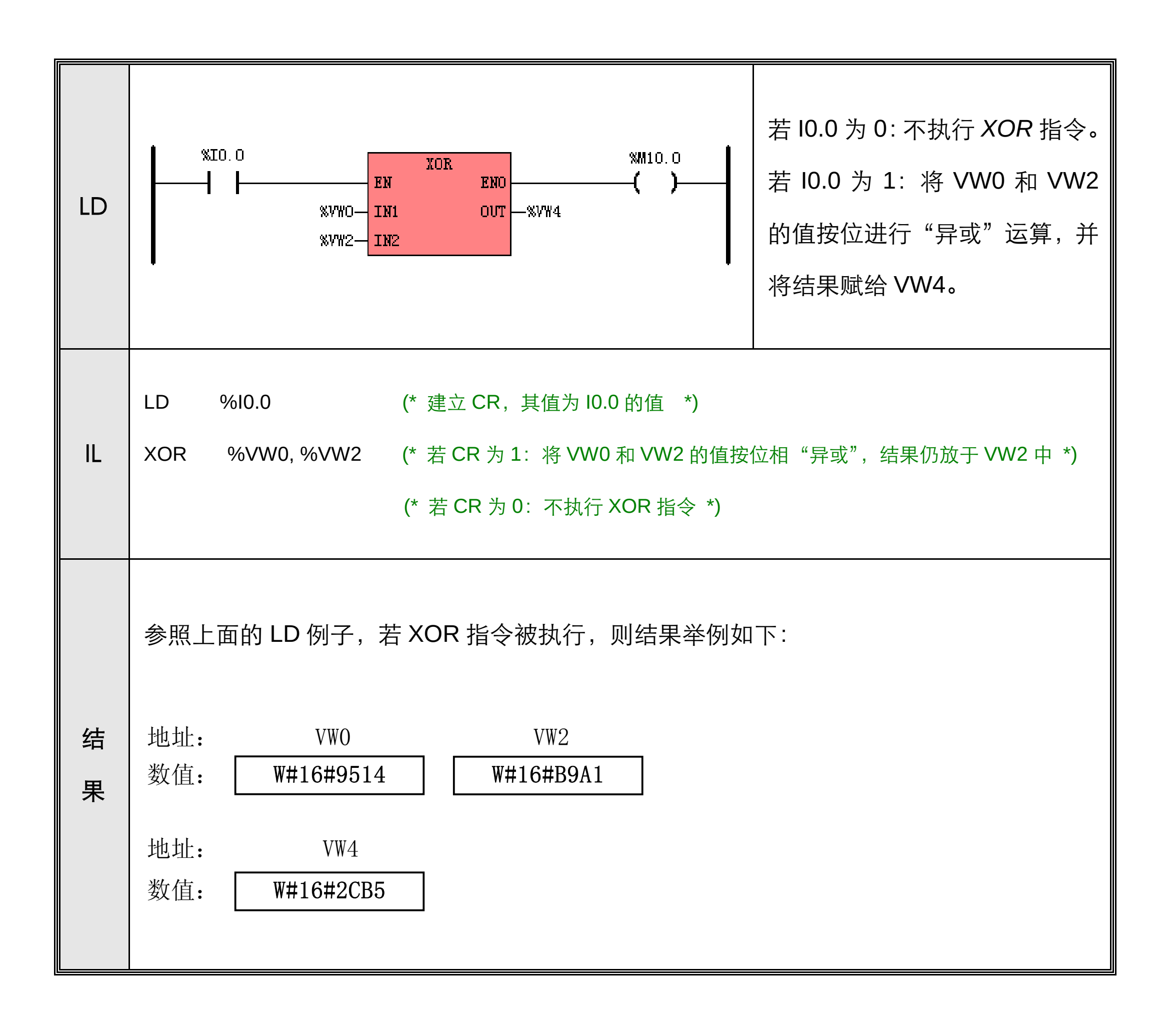
	名称	指令格式	影响 CR 值	
LD	按位异或	XOR EN　ENO IN1　OUT IN2		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	按位异或	XOR　　IN, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM

在 LD 中，*EN* 作为使能端决定了是否执行 *XOR* 指令。如果 *EN* 为 1，则将输入 *IN1 和 IN2* 按二进制位进行“异或”运算后，将结果置于 *OUT* 中。参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *OR* 指令。如果 CR 为 1，那么将参数 *IN 和 OUT* 按二进制位进行“异或”运算后，将结果仍旧置于 *OUT* 中。参数 *IN*、*OUT* 的数据类型必须一致。*OR* 指令的执行不影响 CR 值。

• 指令使用举例



LD	%I0.0 —┤ ├— XOR (EN, ENO); %VW0—IN1, %VW2—IN2, OUT—%VW4; —( )— %M10.0	若 I0.0 为 0: 不执行 *XOR* 指令。若 I0.0 为 1：将 VW0 和 VW2 的值按位进行“异或”运算，并将结果赋给 VW4。
IL	LD　%I0.0　(* 建立 CR，其值为 I0.0 的值　*) XOR　%VW0, %VW2　(* 若 CR 为 1：将 VW0 和 VW2 的值按位相“异或”，结果仍放于 VW2 中 *) (* 若 CR 为 0：不执行 XOR 指令 *)	
结果	参照上面的 LD 例子，若 XOR 指令被执行，则结果举例如下： 地址：VW0　VW2 数值：W#16#9514　W#16#B9A1 地址：VW4 数值：W#16#2CB5	

6.6 移位指令

6.6.1 SHL（左移）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	左移	SHL（EN、IN、N → ENO、OUT）		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	左移	SHL OUT, N	U	

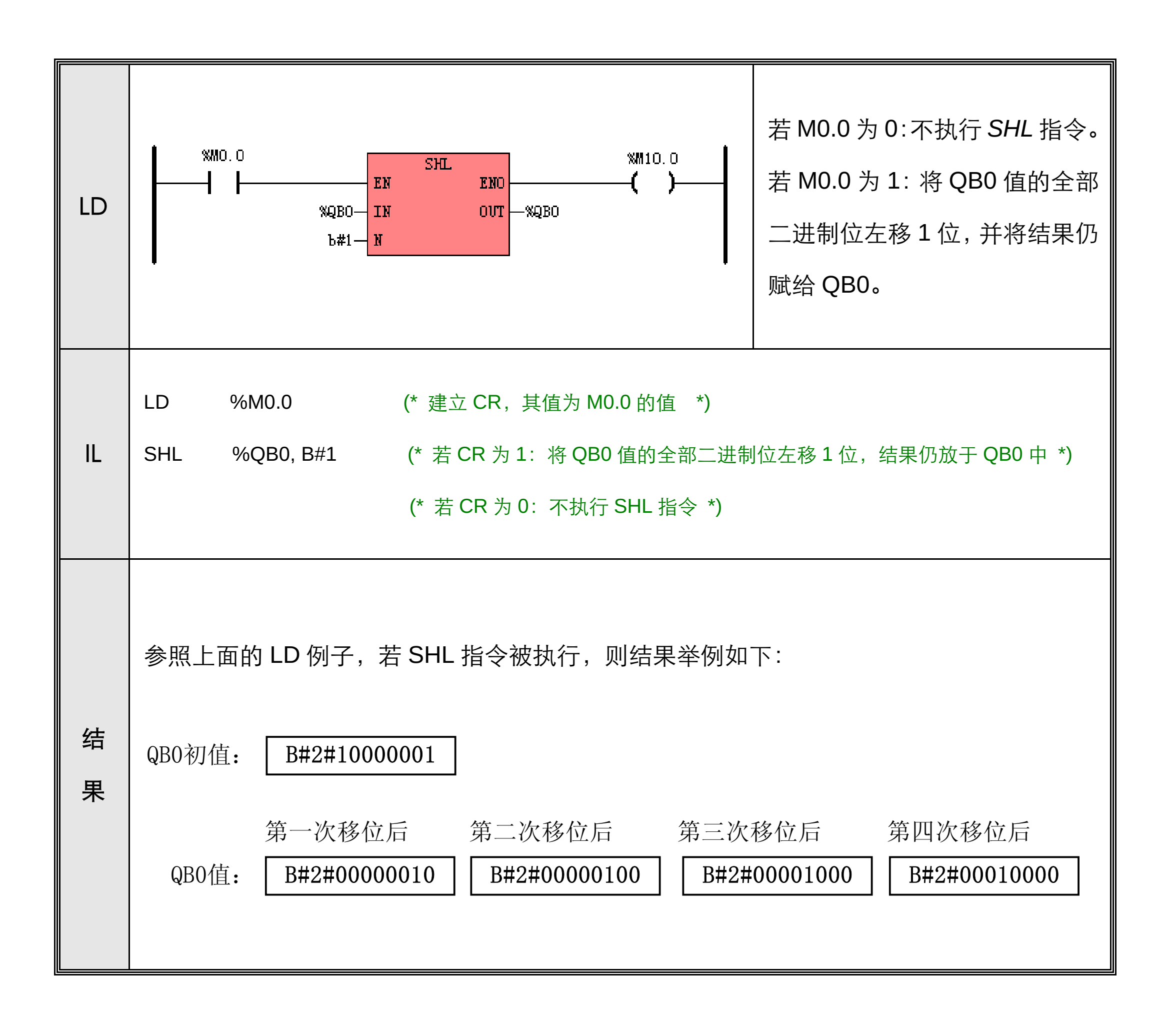
参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
N	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM

在 LD 中， *EN* 作为使能端决定了是否执行 *SHL* 指令。如果 *EN* 为 1，那么将输入 *IN* 的全部二进制位向左移动 *N* 位，同时移出的高位被舍弃并且低位补 0，最终的结果置于 *OUT* 中。参数 *IN*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *SHL* 指令。如果 CR 值为 1，那么将参数 *OUT* 的全部二进制位向左移动 *N* 位，同时移出的高位被舍弃并且低位补 0，最终的结果仍旧被置于 *OUT* 中。*SHL* 指令的执行不影响 CR 值。

N 的最大有效值是参数 IN、OUT 数据类型的位长度。

• 指令使用举例



LD	(梯形图)	若 M0.0 为 0:不执行 *SHL* 指令。 若 M0.0 为 1：将 QB0 值的全部二进制位左移 1 位，并将结果仍赋给 QB0。
IL	LD　%M0.0　(* 建立 CR，其值为 M0.0 的值 *) SHL　%QB0, B#1　(* 若 CR 为 1：将 QB0 值的全部二进制位左移 1 位，结果仍放于 QB0 中 *) (* 若 CR 为 0：不执行 SHL 指令 *)	
结果	参照上面的 LD 例子，若 SHL 指令被执行，则结果举例如下： QB0初值： B#2#10000001 QB0值：第一次移位后 B#2#00000010；第二次移位后 B#2#00000100；第三次移位后 B#2#00001000；第四次移位后 B#2#00010000	

6.6.2　ROL（循环左移）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	循环左移	ROL: EN, IN, N / ENO, OUT		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	循环左移	ROL　　OUT, N	U	

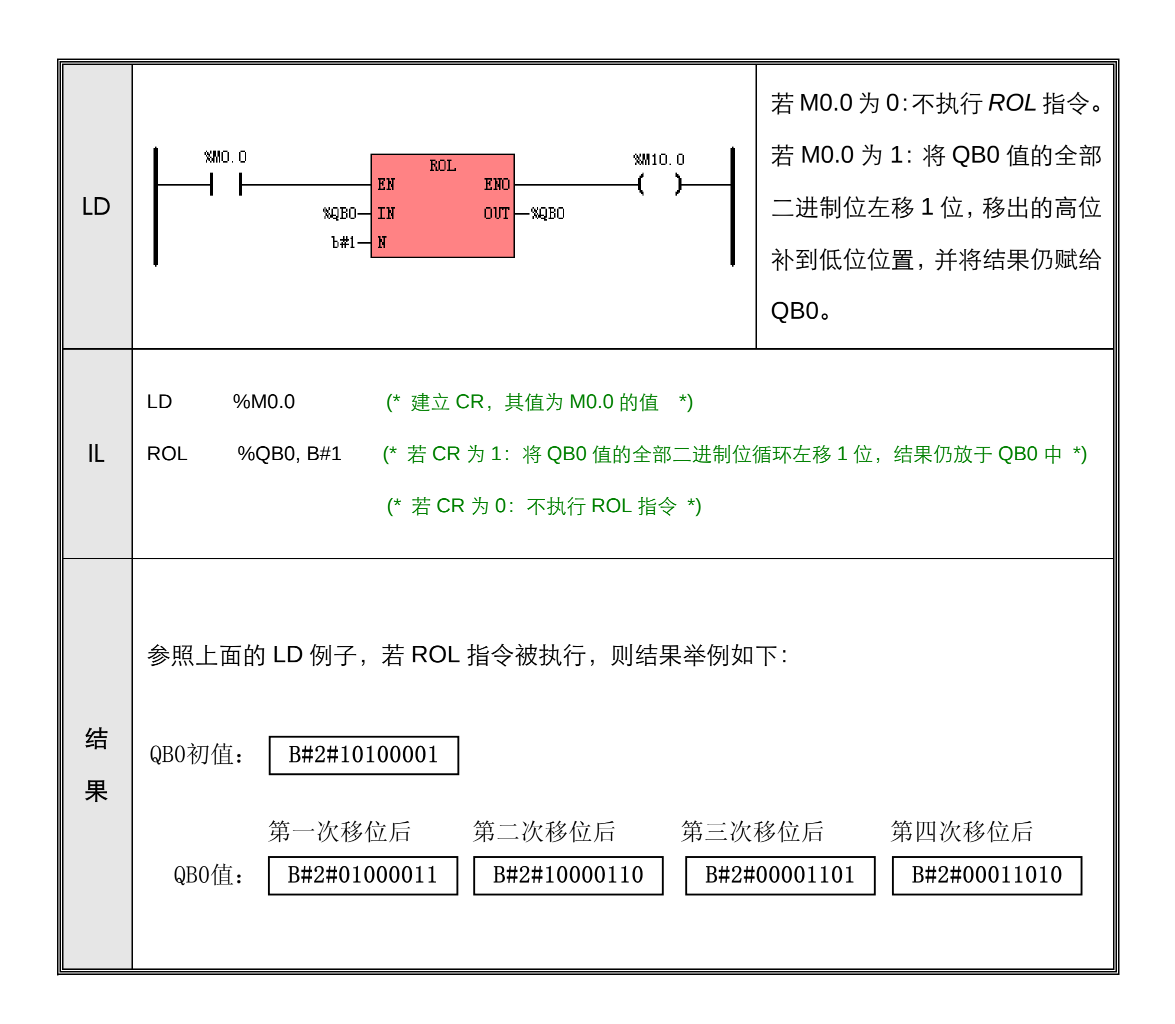
参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
N	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM

在 LD 中，*EN* 作为使能端决定了是否执行 *ROL* 指令。如果 *EN* 为 1，那么将输入 *IN* 的全部二进制位向左移动 *N* 位，同时移出的高位被依次移进低位位置，最终的结果置于 *OUT* 中。参数 *IN*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *ROL* 指令。如果 CR 值为 1，那么将参数 *OUT* 的全部二进制位向左移动 *N* 位，同时移出的高位被依次移进低位位置，最终的结果仍旧被置于 *OUT* 中。*ROL* 指令的执行不影响 CR 值。

N 的最大有效值是参数 IN、OUT 数据类型的位长度。

• 指令使用举例



LD		若 M0.0 为 0：不执行 *ROL* 指令。若 M0.0 为 1：将 QB0 值的全部二进制位左移 1 位，移出的高位补到低位位置，并将结果仍赋给 QB0。
IL	LD　　%M0.0　　(* 建立 CR，其值为 M0.0 的值 *) ROL　　%QB0, B#1　　(* 若 CR 为 1：将 QB0 值的全部二进制位循环左移 1 位，结果仍放于 QB0 中 *) (* 若 CR 为 0：不执行 ROL 指令 *)	
结果	参照上面的 LD 例子，若 ROL 指令被执行，则结果举例如下： QB0初值：B#2#10100001 第一次移位后 / 第二次移位后 / 第三次移位后 / 第四次移位后 QB0值：B#2#01000011　B#2#10000110　B#2#00001101　B#2#00011010	

6.6.3 SHR（右移）

• 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	右移	SHR: EN, IN, N / ENO, OUT		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	右移	SHR OUT, N	U	

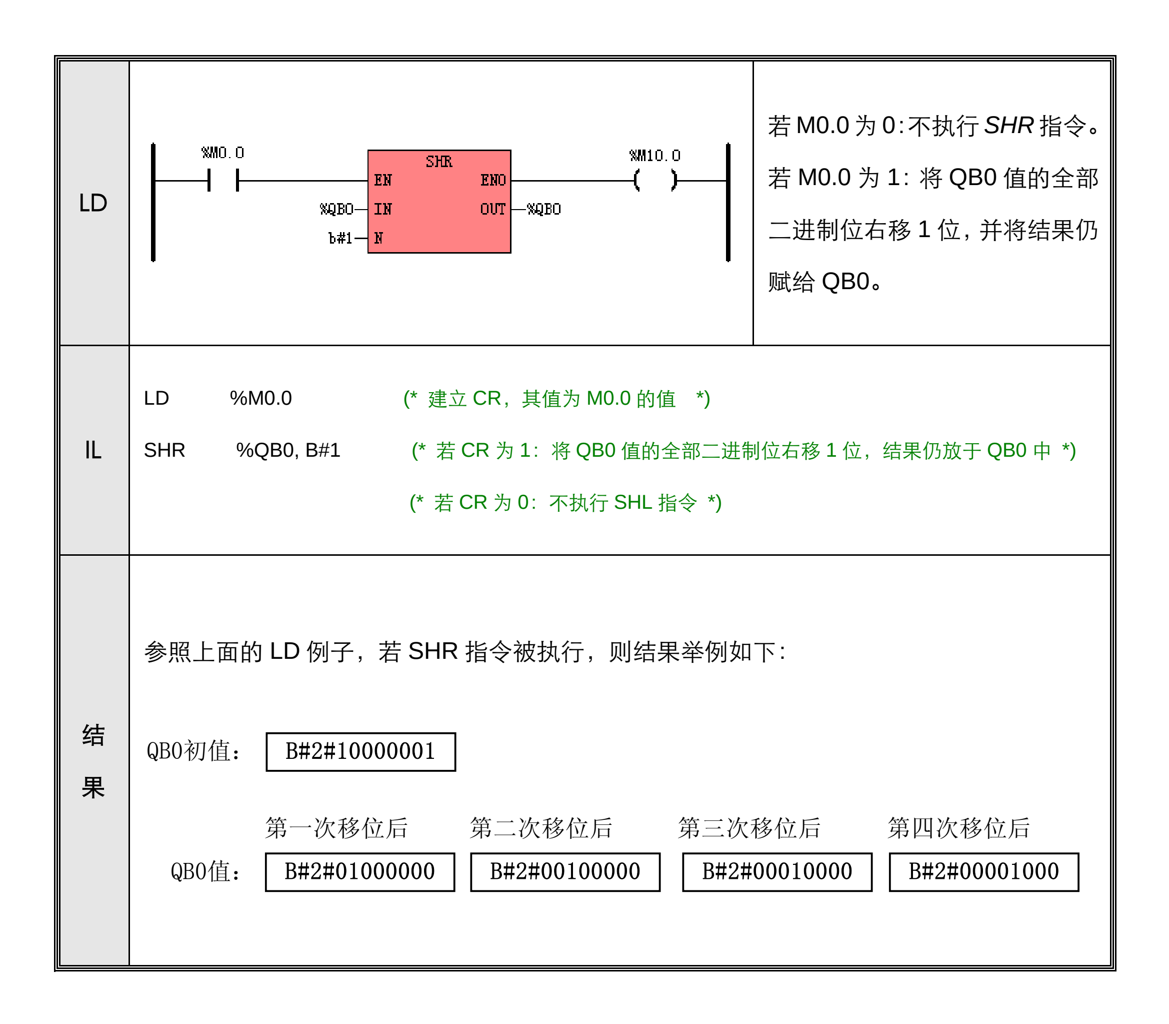
参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
N	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM

在 LD 中， *EN* 作为使能端决定了是否执行 *SHR* 指令。如果 *EN* 为 1，那么将输入 *IN* 的全部二进制位向右移动 *N* 位，同时移出的低位被舍弃并且高位补 0，最终的结果置于 *OUT* 中。参数 *IN*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *SHR* 指令。如果 CR 值为 1，那么将参数 *OUT* 的全部二进制位向右移动 *N* 位，同时移出的低位被舍弃并且高位补 0，最终的结果仍旧被置于 *OUT* 中。*SHR* 指令的执行不影响 CR 值。

参数 N 最大的有效值是参数 IN、OUT 数据类型的位长度。

• 指令使用举例



LD	(ladder diagram)	若 M0.0 为 0:不执行 *SHR* 指令。 若 M0.0 为 1: 将 QB0 值的全部二进制位右移 1 位，并将结果仍赋给 QB0。
IL	LD　%M0.0　(* 建立 CR，其值为 M0.0 的值 *) SHR　%QB0, B#1　(* 若 CR 为 1: 将 QB0 值的全部二进制位右移 1 位，结果仍放于 QB0 中 *) (* 若 CR 为 0: 不执行 SHL 指令 *)	
结果	参照上面的 LD 例子，若 SHR 指令被执行，则结果举例如下: QB0初值: B#2#10000001 QB0值: 第一次移位后 B#2#01000000；第二次移位后 B#2#00100000；第三次移位后 B#2#00010000；第四次移位后 B#2#00001000	

6.6.4 ROR（循环右移）

• 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ CPU304 ☑ CPU306 ☑ CPU308
LD	循环右移	ROR EN ENO IN OUT N		
IL	循环右移	ROR OUT, N	U	

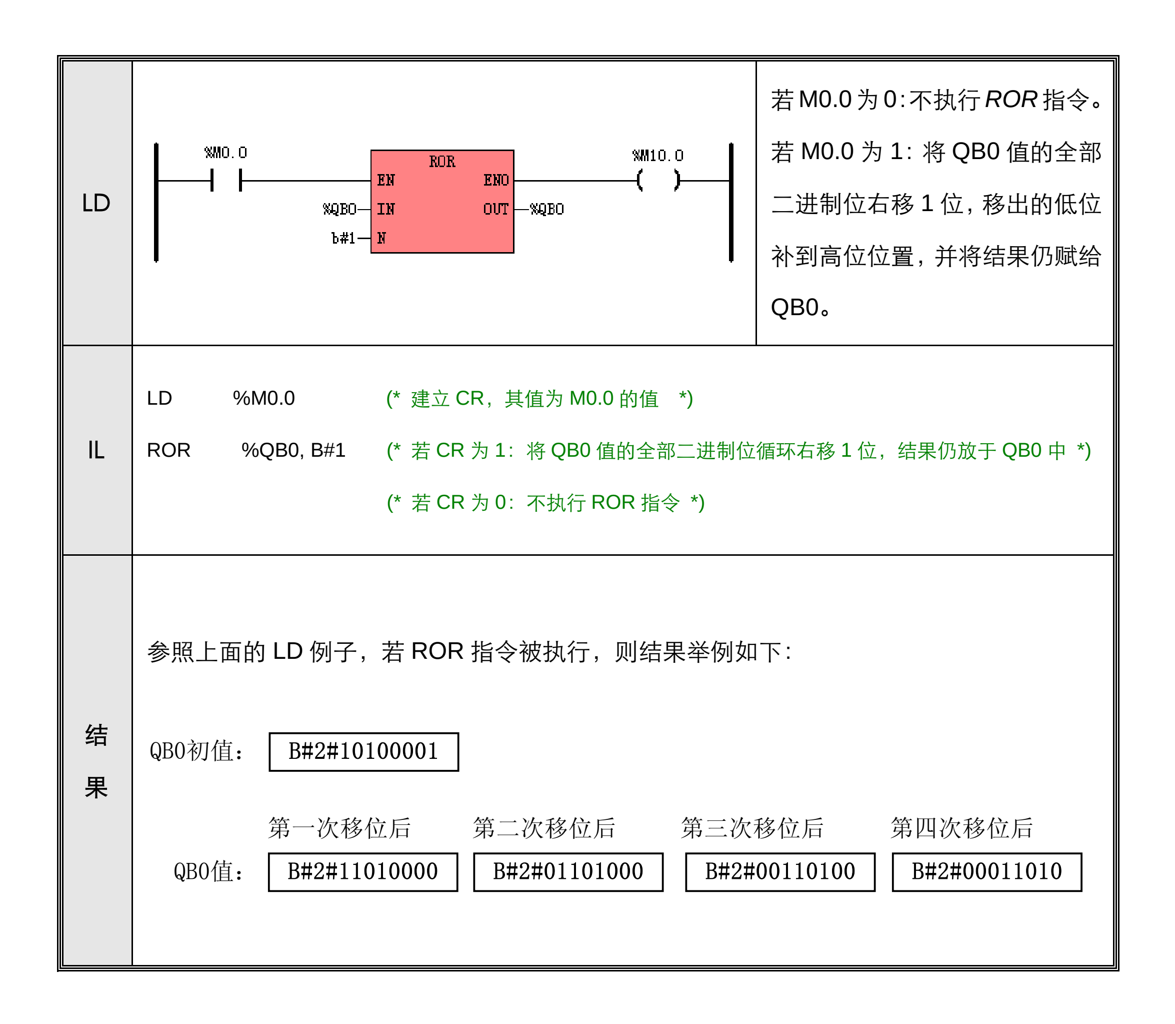
参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量
N	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM

在 LD 中， *EN* 作为使能端决定了是否执行 *ROR* 指令。如果 *EN* 为 1，那么将输入 *IN* 的全部二进制位向右移动 *N* 位，同时移出的低位被依次移进高位位置，最终的结果置于 *OUT* 中。参数 *IN*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *ROR* 指令。如果 CR 值为 1，那么将参数 *OUT* 的全部二进制位向右移动 *N* 位，同时移出的低位被依次移进高位位置，最终的结果仍旧被置于 *OUT* 中。*ROR* 指令的执行不影响 CR 值。

N 的最大有效值是参数 IN、OUT 数据类型的位长度。

- 指令使用举例



<table>
<tr><td>LD</td><td>%M0.0 / ROR / EN / ENO / %QB0 — IN / OUT — %QB0 / b#1 — N / %M10.0</td><td>若 M0.0 为 0:不执行 ROR 指令。
若 M0.0 为 1：将 QB0 值的全部二进制位右移 1 位，移出的低位补到高位位置，并将结果仍赋给 QB0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %M0.0 (* 建立 CR，其值为 M0.0 的值 *)
ROR %QB0, B#1 (* 若 CR 为 1：将 QB0 值的全部二进制位循环右移 1 位，结果仍放于 QB0 中 *)
(* 若 CR 为 0：不执行 ROR 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 ROR 指令被执行，则结果举例如下：
QB0初值： B#2#10100001
第一次移位后 / 第二次移位后 / 第三次移位后 / 第四次移位后
QB0值： B#2#11010000 / B#2#01101000 / B#2#00110100 / B#2#00011010</td></tr>
</table>

6.7 类型转换

6.7.1 DI_TO_R（双整型转实型）

• 指令及其操作数说明

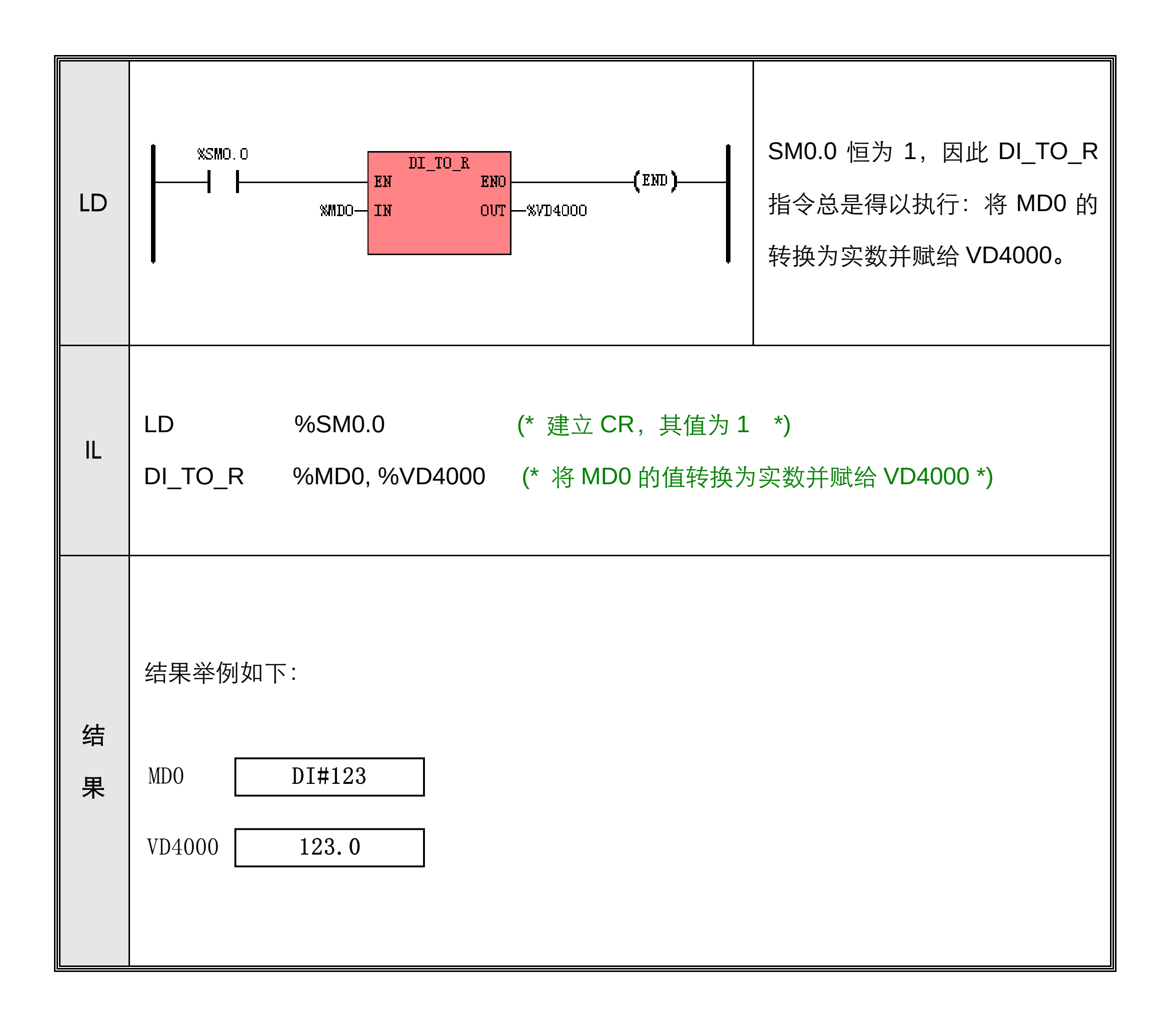
	名称	指令格式	影响 CR 值	
LD	双整型转实型	DI_TO_R EN ENO IN OUT		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	双整型转实型	DI_TO_R IN, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	DINT	I、Q、M、V、L、SM、HC、常量
OUT	输出	REAL	V、L

在 LD 中， *EN* 作为使能端决定了是否执行 *DI_TO_R* 指令。如果 *EN* 为 1，那么将输入 *IN* 从双整型数据转换为实型数据并赋给 *OUT*。

在 IL 中，由 CR 值决定了是否执行 *DI_TO_R* 指令。如果 CR 值为 1，那么将输入 *IN* 从双整型数据转换为实型数据并赋给 *OUT*。*DI_TO_R* 指令的执行不影响 CR 值。

• 指令使用举例



LD	%SM0.0 — DI_TO_R (EN, ENO; %MD0 — IN, OUT — %VD4000) — (END)	SM0.0 恒为 1，因此 DI_TO_R 指令总是得以执行：将 MD0 的转换为实数并赋给 VD4000。
IL	LD　%SM0.0　(* 建立 CR，其值为 1 *) DI_TO_R　%MD0, %VD4000　(* 将 MD0 的值转换为实数并赋给 VD4000 *)	
结果	结果举例如下： MD0　DI#123 VD4000　123.0	

6.7.2 R_TO_DI（实型转双整型）

• 指令及其操作数说明

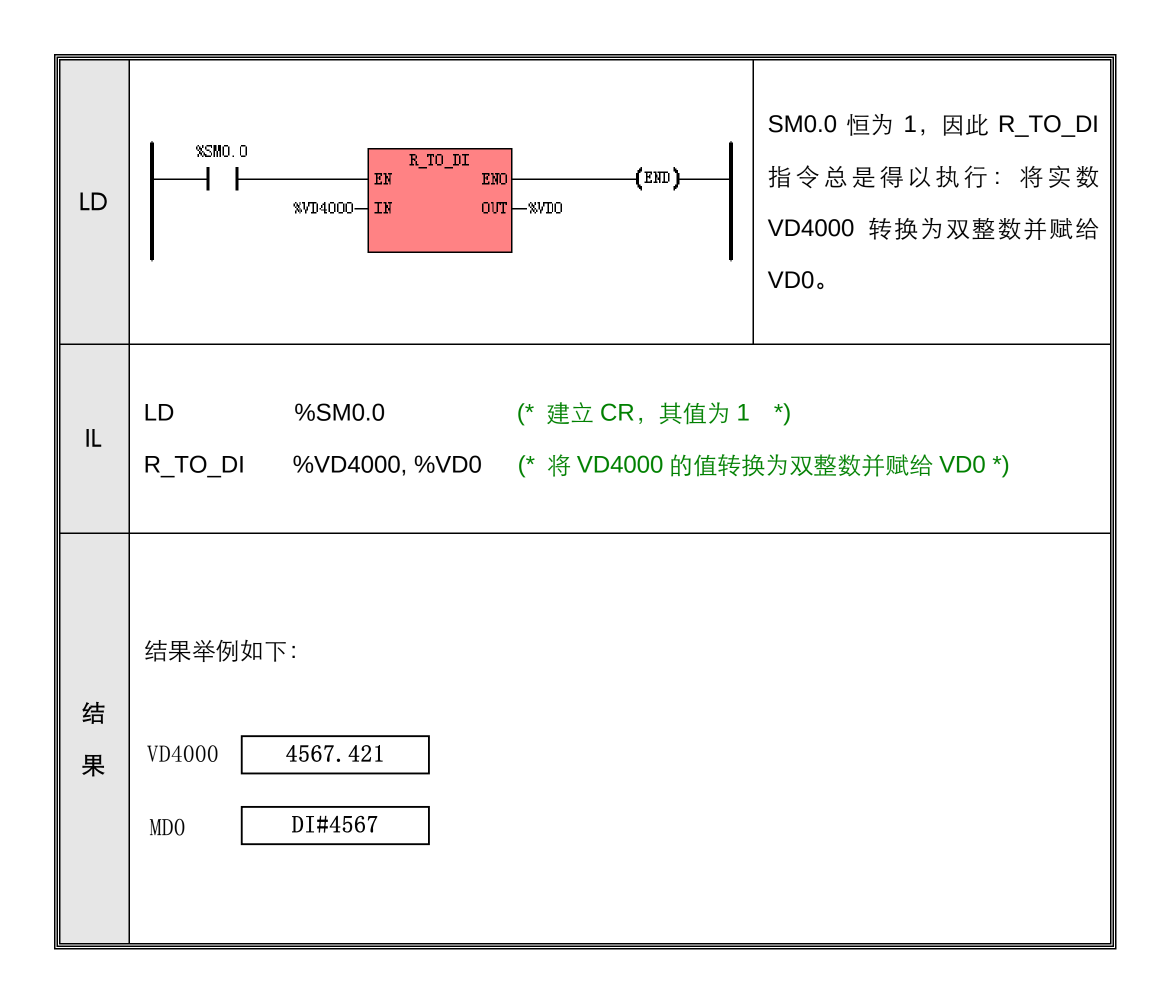
	名称	指令格式	影响 CR 值	☑ CPU304 ☑ CPU306 ☑ CPU308
LD	实型转双整型	R_TO_DI EN ENO IN OUT		
IL	实型转双整型	R_TO_DI IN, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L 、常量
OUT	输出	DINT	M、V、L、SM

在 LD 中，*EN* 作为使能端决定了是否执行 *R_TO_DI* 指令。如果 *EN* 为 1，那么将实数 *IN* 取整转换成双整型数据（舍弃小数部分）并赋给 *OUT*。

在 IL 中，由 CR 值决定了是否执行 *R_TO_DI* 指令。如果 CR 值为 1，那么将实数 *IN* 取整转换成双整型数据（舍弃小数部分）并赋给 *OUT*。*R_TO_DI* 指令的执行不影响 CR 值。

• 指令使用举例



LD	%SM0.0 R_TO_DI EN ENO (END) %VD4000 IN OUT %VD0	SM0.0 恒为 1，因此 R_TO_DI 指令总是得以执行：将实数 VD4000 转换为双整数并赋给 VD0。
IL	LD　%SM0.0　(* 建立 CR，其值为 1 *) R_TO_DI　%VD4000, %VD0　(* 将 VD4000 的值转换为双整数并赋给 VD0 *)	
结果	结果举例如下： VD4000　4567.421 MD0　DI#4567	

6.8 数学运算

6.8.1 ADD（加法）、SUB（减法）

- 指令及其操作数说明

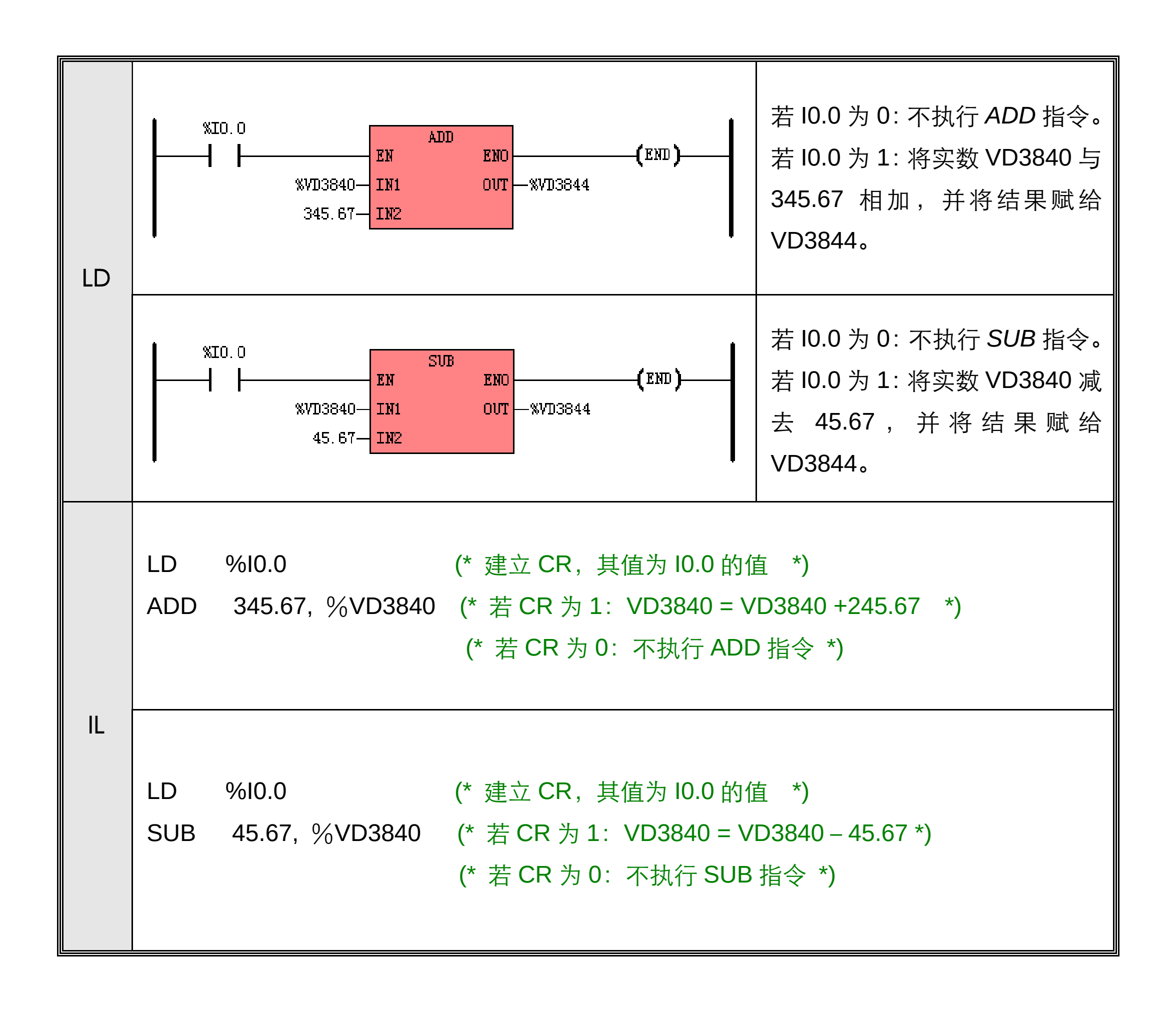
	名称	指令格式	影响 CR 值	
LD	加法	ADD: EN ENO, IN1 OUT, IN2		☑ CPU304 ☑ CPU306 ☑ CPU308
	减法	SUB: EN ENO, IN1 OUT, IN2		
IL	加法	ADD IN1, OUT	U	
	加法	SUB IN1, OUT		

参数	输入/输出	数据类型	允许的内存区
IN1	输入	INT、DINT、REAL	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量
IN2	输入	INT、DINT、REAL	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量
OUT	输出	INT、DINT、REAL	Q、AQ、M、V、L、SM

在 LD 中，*EN* 作为使能端决定了是否执行 *ADD*、*SUB* 指令。若 *EN* 值为 1，则 *ADD* 指令实现的功能是：*OUT=IN1+IN2*；*SUB* 指令实现的功能是：*OUT=IN1 – IN2*。参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *ADD*、*SUB* 指令。如果 CR 值为 1，则 *ADD* 指令实现的功能是：*OUT= OUT + IN1*；*SUB* 指令实现的功能是：*OUT= OUT – IN1*。*ADD*、*SUB* 指令的执行不影响 CR 值。参数 *IN1*、*OUT* 的数据类型必须一致。

- 指令使用举例



LD	（梯形图：ADD）	若 I0.0 为 0：不执行 *ADD* 指令。若 I0.0 为 1：将实数 VD3840 与 345.67 相加，并将结果赋给 VD3844。
	（梯形图：SUB）	若 I0.0 为 0：不执行 *SUB* 指令。若 I0.0 为 1：将实数 VD3840 减去 45.67，并将结果赋给 VD3844。
IL	LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *) ADD 345.67，%VD3840 (* 若 CR 为 1：VD3840 = VD3840 +245.67 *) (* 若 CR 为 0：不执行 ADD 指令 *)	
	LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *) SUB 45.67，%VD3840 (* 若 CR 为 1：VD3840 = VD3840 – 45.67 *) (* 若 CR 为 0：不执行 SUB 指令 *)	

6.8.2　MUL（乘法）、DIV（除法）

- 指令及其操作数说明

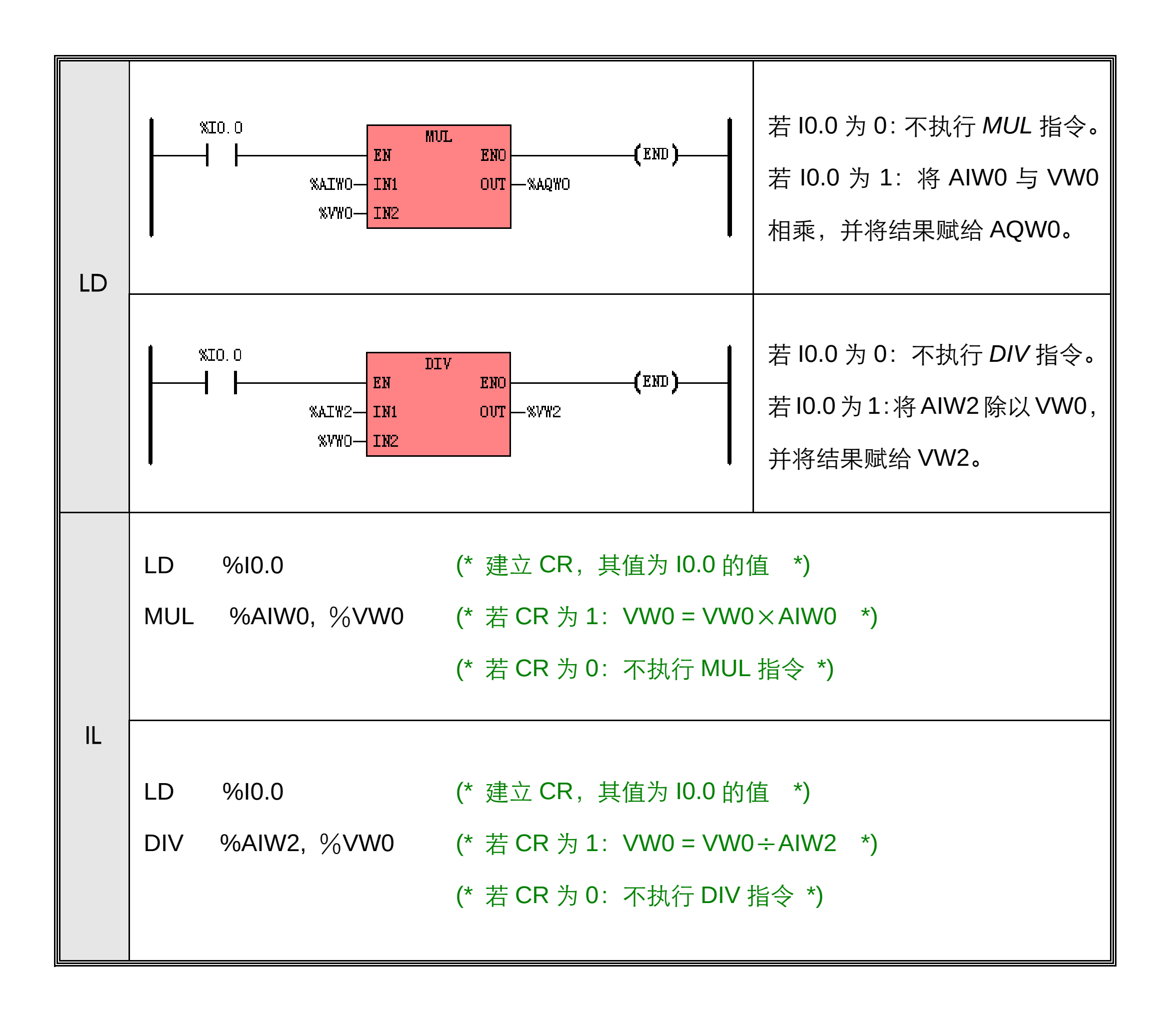
	名称	指令格式	影响 CR 值	
LD	乘法	MUL EN ENO IN1 OUT IN2		☑ CPU304 ☑ CPU306 ☑ CPU308
	除法	DIV EN ENO IN1 OUT IN2		
IL	乘法	MUL　IN1, OUT	U	
	除法	DIV　IN1, OUT		

参数	输入/输出	数据类型	允许的内存区
IN1	输入	INT、DINT、REAL	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量
IN2	输入	INT、DINT、REAL	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量
OUT	输出	INT、DINT、REAL	Q、AQ、M、V、L、SM

在 LD 中，*EN* 作为使能端决定了是否执行 *MUL*、*DIV* 指令。若 *EN* 值为 1，则 *MUL* 指令实现的功能是：*OUT=IN1×IN2*；*DIV* 指令实现的功能是：*OUT=IN1÷IN2*。参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *MUL*、*DIV* 指令。如果 CR 值为 1，则 *MUL* 指令实现的功能是：*OUT= OUT×IN1*；*DIV* 指令实现的功能是：*OUT= OUT÷IN1*。*MUL*、*DIV* 指令的执行不影响 CR 值。参数 *IN1*、*OUT* 的数据类型必须一致。

• 指令使用举例



LD	%I0.0 —| |— MUL (EN, ENO; %AIW0—IN1, OUT—%AQW0; %VW0—IN2) —(END)	若 I0.0 为 0: 不执行 *MUL* 指令。 若 I0.0 为 1：将 AIW0 与 VW0 相乘，并将结果赋给 AQW0。
	%I0.0 —| |— DIV (EN, ENO; %AIW2—IN1, OUT—%VW2; %VW0—IN2) —(END)	若 I0.0 为 0：不执行 *DIV* 指令。 若 I0.0 为 1:将 AIW2 除以 VW0，并将结果赋给 VW2。
IL	LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *) MUL %AIW0，%VW0 (* 若 CR 为 1：VW0 = VW0×AIW0 *) (* 若 CR 为 0：不执行 MUL 指令 *)	
	LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *) DIV %AIW2，%VW0 (* 若 CR 为 1：VW0 = VW0÷AIW2 *) (* 若 CR 为 0：不执行 DIV 指令 *)	

6.8.3 MOD（求余数）

- 指令及其操作数说明

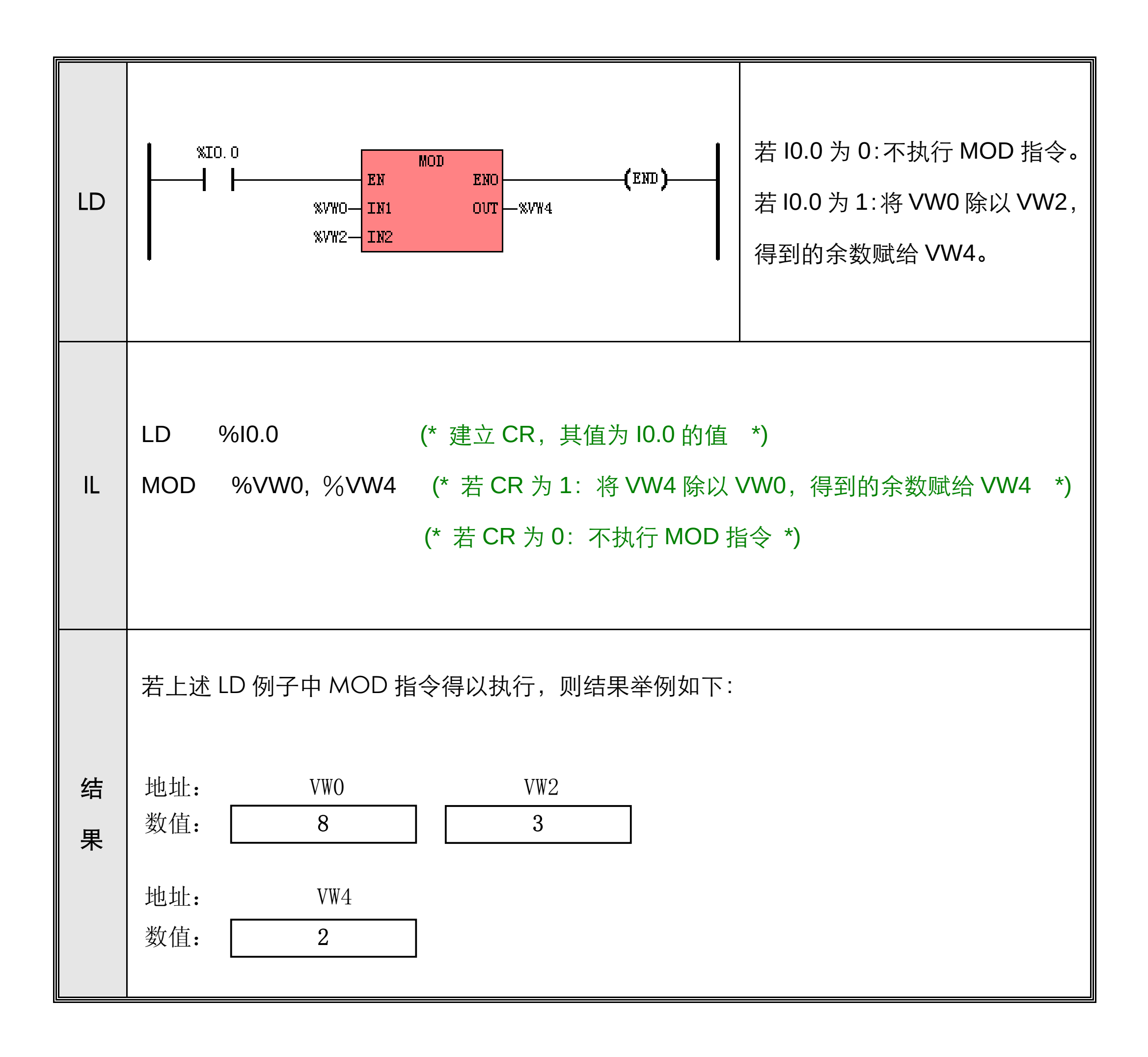
	名称	指令格式	影响 CR 值	
LD	求余数	MOD EN ENO IN1 OUT IN2		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	求余数	MOD IN1, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量
IN2	输入	BYTE、INT、DINT	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量
OUT	输出	BYTE、INT、DINT	Q、AQ、M、V、L、SM

在 LD 中， *EN* 作为使能端决定了是否执行 *MOD* 指令。若 *EN* 值为 1，则将输入 *IN1* 除以 *IN2*，并将所得余数赋给 *OUT*。参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *MUL*、*DIV* 指令。如果 CR 值为 1，则将 *OUT* 除以 *IN1*，并将所得余数赋给 *OUT*。 *MOD* 指令的执行不影响 CR 值。参数 *IN1*、*OUT* 的数据类型必须一致。

• 指令使用举例



LD	%I0.0 MOD EN ENO %VW0 IN1 OUT %VW4 %VW2 IN2 (END)	若 I0.0 为 0:不执行 MOD 指令。 若 I0.0 为 1:将 VW0 除以 VW2，得到的余数赋给 VW4。
IL	LD　　%I0.0　　　(* 建立 CR，其值为 I0.0 的值　*) MOD　　%VW0，%VW4　　(* 若 CR 为 1：将 VW4 除以 VW0，得到的余数赋给 VW4　*) 　　　　(* 若 CR 为 0：不执行 MOD 指令 *)	
结果	若上述 LD 例子中 MOD 指令得以执行，则结果举例如下: 地址：VW0 VW2 数值：8 3 地址：VW4 数值：2	

6.8.4 INC（加 1）、DEC（减 1）

- 指令及其操作数说明

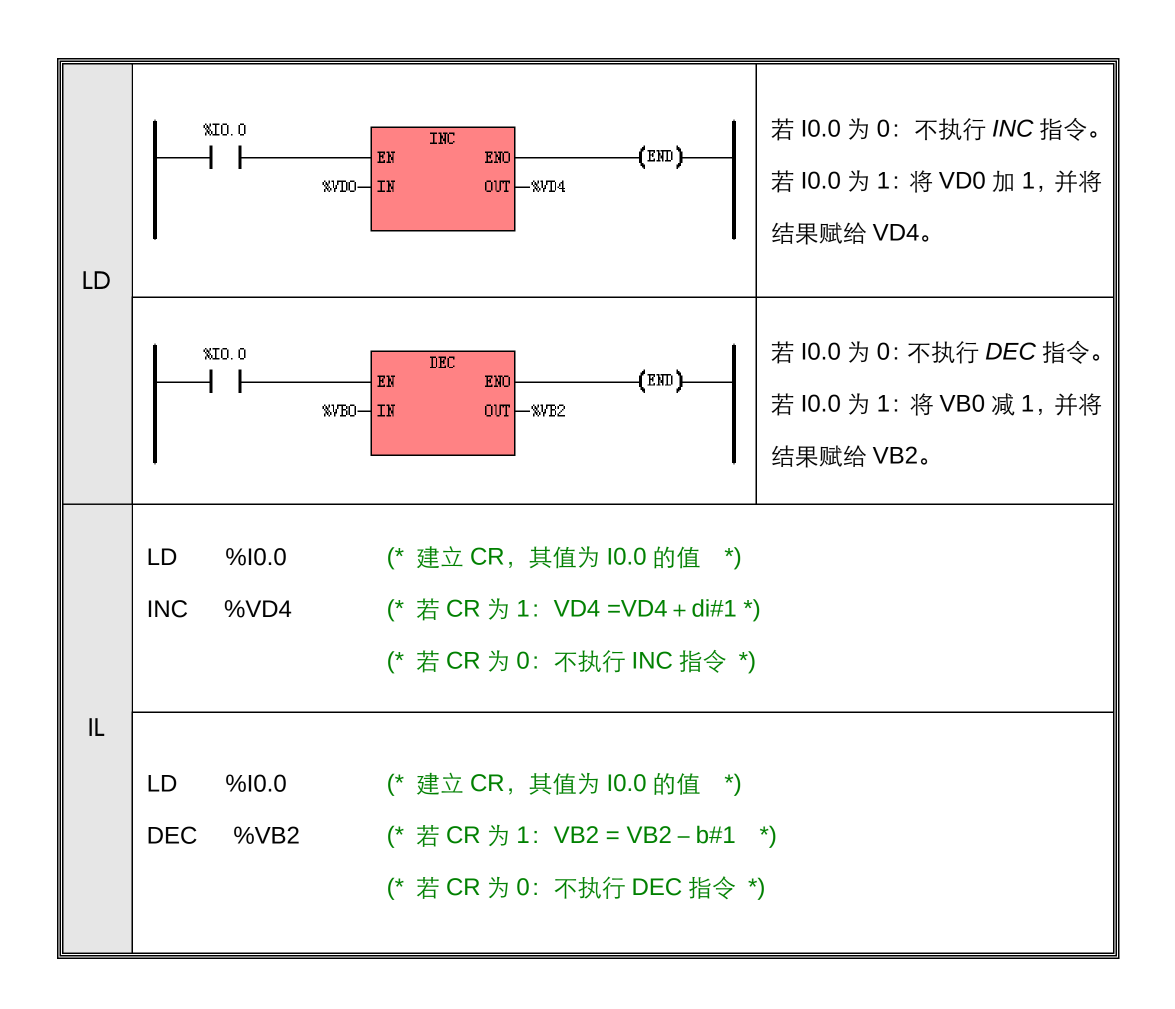
<table>
<tr><th></th><th>名称</th><th>指令格式</th><th>影响 CR 值</th><td rowspan="5">☑ CPU304
☑ CPU306
☑ CPU308</td></tr>
<tr><td rowspan="2">LD</td><td>加 1</td><td>INC
EN ENO
IN OUT</td><td></td></tr>
<tr><td>减 1</td><td>DEC
EN ENO
IN OUT</td><td></td></tr>
<tr><td rowspan="2">IL</td><td>加 1</td><td>INC OUT</td><td rowspan="2">U</td></tr>
<tr><td>减 1</td><td>DEC OUT</td></tr>
</table>

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、INT、DINT	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量
OUT	输出	BYTE、INT、DINT	Q、AQ、M、V、L、SM

在 LD 中， *EN* 作为使能端决定了是否执行 *INC*、*DEC* 指令。若 *EN* 值为 1，则 *INC* 指令实现的功能是：*OUT=IN＋1*；*DEC* 指令实现的功能是：*OUT=IN－1*。参数 *IN*、*OUT* 的数据类型必须一致。

在 IL 中，由 CR 值决定了是否执行 *INC*、*DEC* 指令。如果 CR 值为 1，则 *INC* 指令实现的功能是：*OUT=OUT＋1*；*DEC* 指令实现的功能是：*OUT=OUT－1*。*INC*、*DEC* 指令的执行不影响 CR 值。

• 指令使用举例



LD	(INC ladder diagram)	若 I0.0 为 0：不执行 *INC* 指令。 若 I0.0 为 1：将 VD0 加 1，并将结果赋给 VD4。
	(DEC ladder diagram)	若 I0.0 为 0：不执行 *DEC* 指令。 若 I0.0 为 1：将 VB0 减 1，并将结果赋给 VB2。
IL	LD　%I0.0　(* 建立 CR，其值为 I0.0 的值 *) INC　%VD4　(* 若 CR 为 1：VD4 =VD4 + di#1 *) (* 若 CR 为 0：不执行 INC 指令 *)	
	LD　%I0.0　(* 建立 CR，其值为 I0.0 的值 *) DEC　%VB2　(* 若 CR 为 1：VB2 = VB2 – b#1 *) (* 若 CR 为 0：不执行 DEC 指令 *)	

6.9 程序控制

在 IL 中，下述跳转指令、返回指令的执行均不影响当时的 CR 值，因此用户必要时应该注意在跳转的目的标号或者程序的返回点之后重新建立新的 CR 值，以免程序执行出现错误。

6.9.1 标号及跳转指令

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	标号	lbl —(LBL)—		☑ CPU304 ☑ CPU306 ☑ CPU308
	无条件跳转	lbl —(JMP)—		
	条件跳转	lbl —(JMPC)—		
	条件取反跳转	lbl —(JMPCN)—		
IL	标号	lbl:	U	
	无条件跳转	JMP lbl		
	条件跳转	JMPC lbl		
	条件取反跳转	JMPCN lbl		

参数	描 述
lbl	合法的标识符

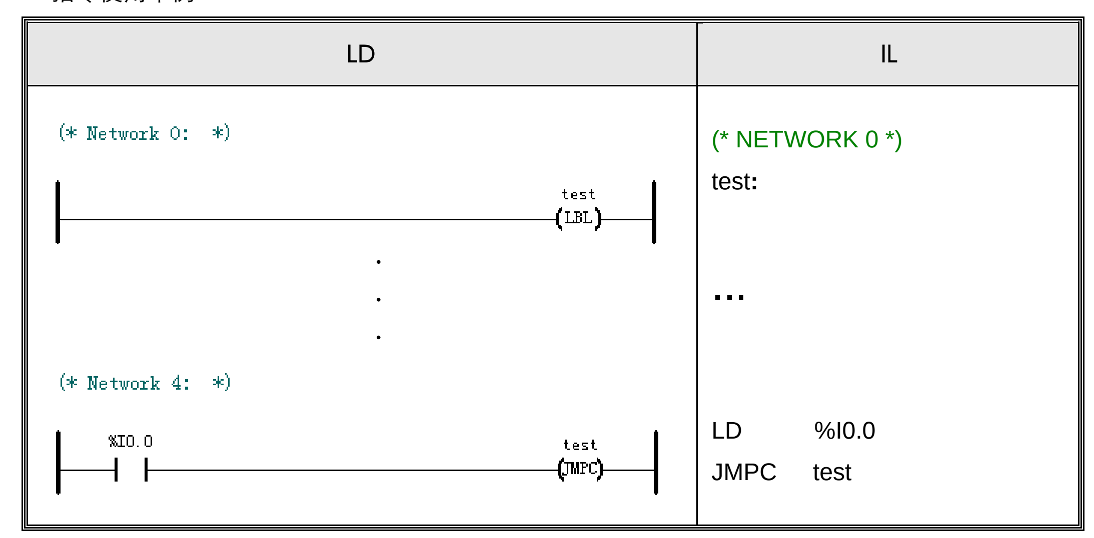
在 LD 中，*LBL* 指令用于在当前位置定义一个标号，该标号将作为跳转指令的目的地。跳转指令是无条件执行的，不建议用户在 *LBL* 指令的左端加入任何元件。实际上，在编译时编译器会将 *LBL* 指令的左端的所有元件忽略掉。标号不允许重复定义。

在 IL 中，标号的定义格式是：***合法的标识符：*** 。标号的定义占用独立的一行。标号不允许重复定义。

在 LD 中，*JMP* 指令的作用是：无条件地将程序跳转到 *lbl* 指定的位置执行。*JMPC* 的作用是：若指令左侧能量流的值为 1，则将程序跳转到 *lbl* 指定的位置执行，否则程序继续依次向下执行。*JMPCN* 指令的作用是：若指令左侧能量流的值为 0，则将程序跳转到 *lbl* 指定的位置执行，否则程序继续依次向下执行。在程序中，跳转指令指定的 *lbl* 标号必须存在。

在 IL 中，*JMP* 指令无条件地将程序跳转到 *lbl* 指定的位置执行。*JMPC* 指令的作用是：若当前的 CR 值为 1，则程序跳转到 *lbl* 指定的位置执行，否则程序继续依次向下执行。*JMPCN* 指令的作用是：若当前的 CR 值为 0，则程序跳转到 *lbl* 指定的位置执行，否则程序继续依次向下执行。在程序中，跳转指令指定的 *lbl* 标号必须存在。

- 指令使用举例



LD	IL
(* Network 0: *) test (LBL) . . . (* Network 4: *) %I0.0 test (JMPC)	(* NETWORK 0 *) test: … LD　%I0.0 JMPC　test

6.9.2　返回指令

返回指令只能在子程序和中断服务程序中使用。

• 指令及其操作数说明

<table>
<tr><th></th><th>名称</th><th>指令格式</th><th>影响 CR 值</th><td rowspan="5">☑ CPU304
☑ CPU306
☑ CPU308</td></tr>
<tr><td rowspan="2">LD</td><td>条件返回</td><td>—(RETC)—</td><td rowspan="2"></td></tr>
<tr><td>条件取反返回</td><td>—(RETCN)—</td></tr>
<tr><td rowspan="2">IL</td><td>条件返回</td><td>RETC</td><td rowspan="2">U</td></tr>
<tr><td>条件取反返回</td><td>RETCN</td></tr>
</table>

在 LD 中，*RETC* 指令的作用是：若指令左侧能量流的值为 1，则中断当前程序并返回到该程序的调用点继续执行，否则该指令不起作用，程序继续向下依次执行。*RETCN* 的作用是：若指令左侧能量流的值为 0，则中断当前程序并返回到该程序的调用点继续执行，否则该指令不起作用。

在 IL 中，*RETC* 指令的作用是：若当前的 CR 值为 1，则中断当前程序并返回到该程序的调用点继续执行，否则该指令不起作用，程序继续向下依次执行。*RETCN* 指令的作用是：若当前的 CR 值为 0，则中断当前程序并返回到该程序的调用点继续执行，否则该指令不起作用。

• 指令使用举例

<table>
<tr>
<td>LD</td>
<td>
主程序：
(* Network 0: *)
%SM0.0 — SBR_0 (EN ENO) — (END)
(* Network 1: *)
%SM0.0 — SBR_1 (EN ENO) — (END)

子程序 SBR_0：
(* Network 0: *)
%I0.0 — (RETC)
(* Network 1: *)
%I0.1 — %I0.2 (/) — %Q0.0 ( )
</td>
<td>
在子程序 SBR_0 中：
若 I0.0 为 0，则继续依次执行下面的指令。
若 I0.0 为 1，则返回到主程序中 SBR_0 的调用点处继续向下执行 Network 1 中的指令。
</td>
</tr>
<tr>
<td>IL</td>
<td colspan="2">
主程序：
LD　%SM0.0　(* 建立 CR，其值恒为 I　*)
CAL　SBR_0　(* 调用子程序 SBR_0 *)
CAL　SBR_1　(* 调用子程序 SBR_1 *)

子程序 SBR_0：
LD　%I0.0　(* 建立 CR，其值为 I0.0 的值　*)
RETC　(* 若 CR 为 1：则返回到主程序中并继续执行 CAL　SBR_1 命令 *)
LD　%I0.1　(* 若 RETC 指令没起作用，则该指令及后续指令得以执行　*)
ANDN　%I0.2
ST　%Q0.0
</td>
</tr>
</table>

6.10 中断指令

6.10.1 ENI（允许中断）、DISI（禁止中断）

• 指令及其操作数说明

<table>
<tr><th></th><th>名称</th><th>指令格式</th><th>影响 CR 值</th><td rowspan="5">☑ CPU304
☑ CPU306
☑ CPU308</td></tr>
<tr><td rowspan="2">LD</td><td>允许中断</td><td>—(ENI)—</td><td rowspan="2"></td></tr>
<tr><td>禁止中断</td><td>—(DISI)—</td></tr>
<tr><td rowspan="2">IL</td><td>允许中断</td><td>ENI</td><td rowspan="2">U</td></tr>
<tr><td>禁止中断</td><td>DISI</td></tr>
</table>

在 LD 中，*ENI* 指令的作用是：若指令左端能量流的值为 1，则允许程序处理中断事件（即当中断发生时 CPU 允许调用中断服务程序），否则不执行该指令。*DISI* 指令的作用是：若指令左端能量流的值为 1，则禁止程序处理中断事件，否则不执行该指令。

在 IL 中，*ENI* 指令的作用是：若当前的 CR 值为 1，则允许程序处理中断事件，否则不执行该指令。*DISI* 指令的作用是：若当前的 CR 值为 1，则禁止程序处理中断事件，否则不执行该指令。*ENI*、*DISI* 指令的执行均不影响当前的 CR 值。

ENI、*DISI* 指令在程序中执行一次即可。

ENI、*DISI* 指令的示例请参见下一节中的指令使用举例。

6.10.1 ATCH（中断连接）、DTCH（中断分离）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	中断连接	ATCH EN ENO INT EVENT		☑ CPU304 ☑ CPU306 ☑ CPU308
	中断分离	DTCH EN ENO EVENT		
IL	中断连接	ATCH INT, EVENT	U	
	中断分离	DTCH EVENT		

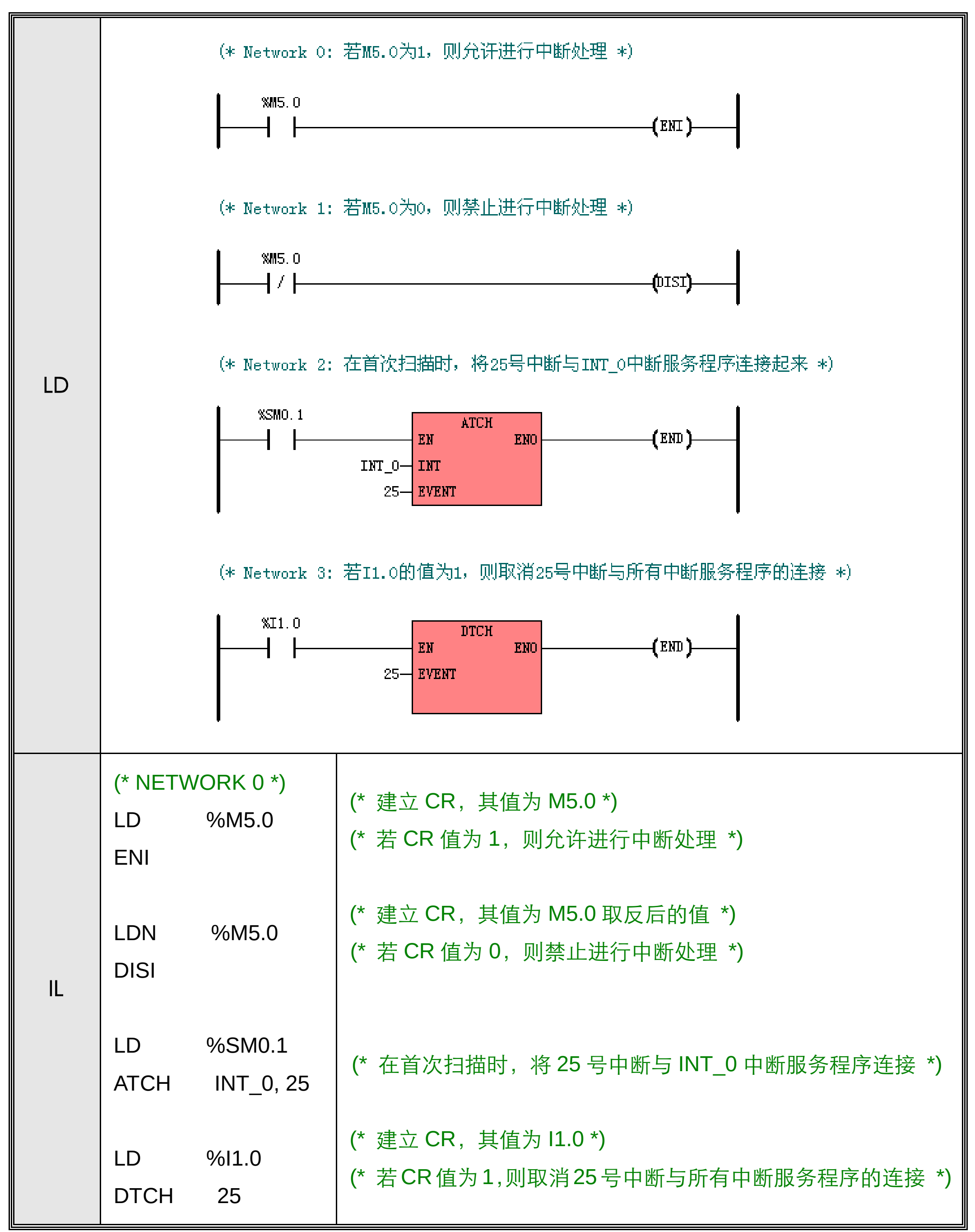
参数	输入/输出	数据类型	参数描述
INT	输入		用户工程中已存在的中断服务程序的名称
EVENT	输入	INT 型常量	中断事件号

在 LD 中，*EN* 作为使能端决定了是否执行 *ATCH*、*DTCH* 指令。若 *EN* 值为 1，则 *ATCH* 指令实现的功能是：将参数 *EVENT* 指定的中断事件与参数 INT 指定的中断服务程序连接起来，这样若 CPU 允许处理中断事件，则当该中断事件发生时，将自动调用该中断服务程序；*DTCH* 指令实现的功能是：取消参数 *EVENT* 指定的中断事件与所有中断服务程序的连接。

在 IL 中，由 CR 值决定了是否执行 *ATCH*、*DTCH* 指令。如果 CR 值为 1，则 *ATCH*、*DTCH* 指令被执行。*ATCH*、*DTCH* 指令的执行不影响 CR 值。

在前文中已经提到：多个中断事件可以调用同一个中断服务程序，但一个中断事件不允许与多个中断服务程序连接。

- 指令使用举例



IL	(* NETWORK 0 *) LD %M5.0 ENI	(* 建立 CR，其值为 M5.0 *) (* 若 CR 值为 1，则允许进行中断处理 *)
	LDN %M5.0 DISI	(* 建立 CR，其值为 M5.0 取反后的值 *) (* 若 CR 值为 0，则禁止进行中断处理 *)
	LD %SM0.1 ATCH INT_0, 25	(* 在首次扫描时，将 25 号中断与 INT_0 中断服务程序连接 *)
	LD %I1.0 DTCH 25	(* 建立 CR，其值为 I1.0 *) (* 若 CR 值为 1，则取消 25 号中断与所有中断服务程序的连接 *)

6.11 通讯指令

本节介绍的指令是用于自由通讯的。所谓的自由通讯是指对于 CPU 中的通讯模块而言是无协议进行通讯的，也就是说 CPU 不对通讯数据作任何变化而只是负责以 ASCII 的形式进行收发。在这种方式下，用户可以编写自己的程序对通讯过程进行控制。

KDN-K3 系列 PLC 的 CPU 本体集成了 1 或 2 个串口，这些串口默认使用 Modbus RTU 协议进行通讯并作为 Modbus 从站。当执行用户程序中的通讯指令时，自由通讯方式被激活，当通讯指令执行结束后，CPU 又将通讯协议切换到 Modbus RTU 协议而无需用户人工参与。

串行通讯参数（包括波特率、数据位等）是在【PLC 硬件配置】中 CPU 模块的〔通讯设置〕页面上进行设置的。

6.11.1 XMT（发送数据）、RCV（接收数据）

- 指令及其操作数说明

<table>
<tr><th></th><th>名称</th><th>指令格式</th><th>影响 CR 值</th><td rowspan="5">☑ CPU304
☑ CPU306
☑ CPU308</td></tr>
<tr><td rowspan="2">LD</td><td>发送数据</td><td>XMT
EN ENO
TBL
PORT</td><td rowspan="2"></td></tr>
<tr><td>接收数据</td><td>RCV
EN ENO
TBL
PORT</td></tr>
<tr><td rowspan="2">IL</td><td>发送数据</td><td>XMT TBL, PORT</td><td rowspan="2">U</td></tr>
<tr><td>接收数据</td><td>RCV TBL, PORT</td></tr>
</table>

参数	输入/输出	数据类型	允许使用的内存区
TBL	输入	BYTE	I、Q、M、V、L、SM
PORT	输入	INT	常量（0 或 1）

XMT 指令用于发送存放在数据缓冲区中的数据。若 SM87.1=1，则 CPU 在发送完缓冲区中的最后一个字符时就会自动产生一个发送完成中断（对于 PORT 0 中断事件号为 30，对于 PORT 1 中断事件号为 32）。参数 *PORT* 定义了所用的通讯口。参数 *TBL* 定义了一个数据缓冲区，缓冲区的第一个字节中存放着用户指定本次将要发送的字节数（1--255），后边依次存放着待发送的数据字节。若发送字节数被设置为 0，则 *XMT* 指令不执行任何操作，当然也不会产生中断。

RCV 指令用于接收数据并将接收到的数据存放在数据缓冲区中。用户在调用 *RCV* 指令后，必须指定有效数据接收的起始条件和结束条件（见下面控制字的定义）。若 SM87.1=1，则 CPU 在退出接收后（无论是正常还是异常退出）就会自动产生一个接收完成中断（对于 PORT 0 中断事件号为 29，对于 PORT 1 中断事件号为 31）。参数 *PORT* 定义了所用的通讯口。参数 *TBL* 定义了一个数据缓冲区，缓冲区的第一个字节中存放着本次接收到的字节数，后边依次存放着接收到的有效数据字节。

在 LD 中，*EN* 作为使能端决定了是否执行 *XMT*、*RCV* 指令。

在 IL 中，由 CR 值决定了是否执行 *XMT*、*RCV* 指令。*XMT*、*RCV* 指令的执行不影响 CR 值。

- SM 区中自由通讯的状态字节及控制字节汇总

除了 XMT、RCV 指令外，在 CPU 中同时还设置了多个状态字（或字节）和控制字（或字节）用于自由通讯。用户必须在程序中对这些控制字进行设置来控制通讯。另外，在通讯过程中 CPU 会自动对通讯状态进行检测，并将检测结果设置到指定的状态字，用户可以读取这些状态信息并在程序中进行相应的处理。下面将对这些状态字和控制字进行简要的汇总。

① 接收状态字节 SMB86

位（只读）		值	含 义
PORT 0	PORT 1		
SM86.0		1	接收到的字符存在奇偶校验错误，但不停止接收。
SM86.1		1	终止接收：达到最大接收字节数（见 SMB94）。
SM86.2		1	终止接收：接收一个字符超时（见 SMW92）。
SM86.3		1	终止接收：系统接收超时。
SM86.4		-	保留。

SM86.5		1	终止接收：接收到了用户定义的结束字符（见 SMB89）。
SM86.6		1	终止接收：参数设置错误，无起始条件接收或者无接收结束条件等。
SM86.7		1	终止接收：用户使用了禁止接收命令（见 SM87.7 ）

② 接收控制字节 SMB87

位		值	描 述
PORT 0	PORT 1		
SM87.0		-	保留。
SM87.1		0	当发送或者接收完成时禁止生成相应的中断。
		1	当发送或者接收完成时允许生成相应的中断。
SM87.2		0	忽略 SMW92 中用户定义的接收字符超时值。
		1	使用 SMW92 中用户定义的接收字符超时值。
SM87.3		-	保留。
SM87.4		0	忽略 SMW90 中用户定义的接收准备时间。
		1	使用 SMW90 中用户定义的接收准备时间来作为启动接收的条件。
SM87.5		0	忽略 SMB89 中用户定义的接收结束字符。
		1	使用 SMB89 中用户定义的接收结束字符。
SM87.6		0	忽略 SMB88 中用户定义的接收开始字符。
		1	使用 SMB88 中用户定义的接收开始字符。
SM87.7		0	禁止接收数据。此禁止条件优先于其它所有的接收控制字。
		1	允许接收数据。

① 其它控制字（或字节）

控制字（字节）		描述
PORT 0	PORT 1	
SMB88		用于存放用户定义的接收起始字符。 执行 RCV 指令后，CPU 收到了起始字符就开始进入有效接收状态，之前收到的数据都会被丢弃。CPU 将起始字符作为接收的第一个有效数据。 如果要使本设置值生效，需要将 SM87.6 置为 1。

SMB89		用于存放用户定义的接收结束字符。 CPU 将以该字符作为接收的最后一个字节。收到该字符后，无论其它的结束条件如何 CPU 都会立刻终止接收状态。 如果要使本设置值生效，需要将 SM87.5 置为 1。
SMW90		用于存放用户定义的接收准备时间（范围为 1~60000ms）。 执行 RCV 指令后，经过了该时间值，CPU 将自动进入有效接收状态而不管是否收到起始字符，此后接收到的数据将被认为是有效数据。 如果要使本设置值生效，需要将 SM87.4 置为 1。
SMW92		用于存放用户定义的接收字符超时值（范围为 1~60000ms）。 执行 RCV 指令并进入有效接收状态后，若在此超时时间内没有接收到任何字符，CPU 就会结束接收状态，无论其它的结束条件如何。 如果要使本置值生效，需要将 SM87.2 置为 1。
SMW94		用于存放用户定义的每次接收字符数（1~255）。 CPU 只要接收到了此数量个有效字符，无论其它的结束条件如何，都会马上终止接收状态。本设置值总是有效。 若该值设置为 0，则 RCV 指令将直接退出。

在自由通讯中，另外还有一个默认的系统接收超时，时间为 90 秒，此超时值的作用如下：在执行 RCV 指令后，若在此系统接收超时内串口上没有收到任何数据，则 CPU 将立刻终止接收并退出 RCV 指令；另外，CPU 在进入有效接收状态后（即接收到起始字符或者经过了 SMW90 中定义的接收准备时间后），将优先使用用户在 SMW92 中定义的接收字符超时值，若用户没有定义，则用该系统接收超时值来决定是否终止接收。

- 自由通讯示例

下面将举例说明自由通讯的使用。在示例中，CPU 将接收一串数据，以回车符作为接收结束字符；若接收正常完成后，则把接收到的数据又发送回去并再次启动接收，若是异常退出接收状态（比如通讯错误、接收超时等），则忽略接收到的数据并再次启动接收。

LD

主程序（MAIN）：

(* Network 0: 如下的程序在CPU刚启动时执行一次，用于初始化自由通讯。
首先设定有效接收状态的起始条件和终止条件。 *)

%SM0.1 — MOVE (EN, ENO; b#16#B6 — IN, OUT — %SMB87) — (END)

(* Network 1: 设定接收准备时间为10ms，接收结束字符为回车符（ASCII为13）。 *)

%SM0.1 — MOVE (EN, ENO; 10 — IN, OUT — %SMW90) — MOVE (EN, ENO; b#16#D — IN, OUT — %SMB89) — (END)

(* Network 2: 设定接收字符超时为500ms，设定每次接收的字符数最大为100个。 *)

%SM0.1 — MOVE (EN, ENO; 500 — IN, OUT — %SMW92) — MOVE (EN, ENO; b#100 — IN, OUT — %SMB94) — (END)

(* Network 3: 将EndReceive程序绑定到接收完成中断，将EndSend程序绑定到发送完成中断。 *)

%SM0.1 — ATCH (EN, ENO; EndReceive — INT; 29 — EVENT) — ATCH (EN, ENO; EndSend — INT; 30 — EVENT) — (END)

(* Network 4: 全局启用中断。 *)

%SM0.1 — (ENI)

(* Network 5: 在CPU刚启动时就启动一次接收任务。 *)

%SM0.1 — RCV (EN, ENO; %VB100 — TBL; 0 — PORT) — (END)

LD

EndReceive（INT00），接收完成中断服务程序：

(* Network 0: 若是收到结束字符，则回发收到的数据然后退出本程序。 *)

%SM86.5 — XMT (EN, ENO, TBL: %VB100, PORT: 0) — (RETC)

(* Network 1: 若是异常退出接收，则仅仅重新启动接收任务。 *)

%SM86.6 / %SM86.3 / %SM86.2 / %SM86.1 / %SM86.0 — RCV (EN, ENO, TBL: %VB100, PORT: 0) — (END)

EndSend（INT01），发送完成中断服务程序：

(* Network 0: 在发送数据完成后，重新启动接收任务。 *)

true — RCV (EN, ENO, TBL: %VB100, PORT: 0) — (END)

IL	**主程序（MAIN）：** (* NETWORK 0 *) LD %SM0.1 (* 在 CPU 刚启动时完成下述任务 *) MOVE B#16#B6, %SMB87 (* 设定有效接收状态的起始条件和结束条件 *) MOVE 10, %SMW90 (* 接收准备时间设定为 10ms *) MOVE B#16#D, %SMB89 (* 接收结束字符设定为回车符（ASCII 为 13）*) MOVE 500, %SMW92 (* 接收字符超时设定为 500ms *) MOVE B#100, %SMB94 (* 每次接收的字符数设定为最大 100 个 *) ATCH EndReceive, 29 (* 将 EndReceive 程序绑定到接收完成中断 *) ATCH EndSend, 30 (* 将 EndSend 程序绑定到发送完成中断 *) ENI (* 全局启用中断 *) RCV %VB100, 0 (* 启动一次接收任务 *)
	EndReive（INT00），接收完成中断服务程序： (* NETWORK 0 *) LD %SM86.5 XMT %VB100, 0 (* 收到结束字符则回发收到的数据 *) RETC (* 然后退出本程序 *) (* NETWORK 1 *) LD %SM86.0 OR %SM86.1 OR %SM86.2 OR %SM86.3 OR %SM86.6 RCV %VB100, 0 (*若是异常退出接收状态，则重新启动接收任务*)
	EndSend（INT01），发送完成中断服务程序： (* NETWORK 0 *) LD TRUE RCV %VB100, 0 (*发送完成后重新启动接收任务*)

6.12 计数器

计数器是 IEC61131-3 标准中定义的功能块之一，共有 CTU、CTD 和 CTUD 三种。关于功能块及其实例的使用请参阅 3.6.4 FB 实例存储区的分配 一节。

6.12.1 CTU（增计数器）、CTD（减计数器）

• CTU 及其操作数说明

	名称	指令格式	影响 CR 值	
LD	增计数器	Cx CTU CU Q R CV PV		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	增计数器	CTU Cx, R, PV	P	

参数	输入/输出	数据类型	允许使用的内存区
Cx	-	计数器实例	C
CU	输入	BOOL	能量流
R	输入	BOOL	I、Q、M、V、L、SM、T、C
PV	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
CV	输出	INT	Q、M、V、L、SM、AQ

在 LD 中，*CTU* 用于对 *CU* 输入端的上升沿进行递增（每次递增 1）计数，当实例 *Cx* 的当前计数值 *CV* 大于等于预设值 *PV* 时，其输出 *Q* 及其状态值均被置为 1。若 *R* 输入端为 1，则实例 *Cx* 被复位，其输出 *Q* 及状态值均被置为 0，同时 *CV* 及其当前计数值也被清零。

在 IL 中，*CTU* 用于对当前 CR 值的上升沿进行递增（每次递增 1）计数，当实例 Cx 的当前计数值大于等于预设值 *PV* 时，*Cx* 的状态值被置为 1。输入 *R* 用于复位实例 *Cx*。CPU 每次扫描

CTU 后，CR 值均被设置为实例 *Cx* 的状态值。

若复位输入 *R* 为 0，则当计数器实例 *Cx* 计数到预设值时还将继续计数直至达到并保持在 INT 型数据的最大值（即 32767）。

- CTD 及其操作数说明

	名称	指令格式	影响 CR 值	
LD	减计数器	Cx CTD CD Q LD CV PV		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	减计数器	CTD Cx, LD, PV	P	

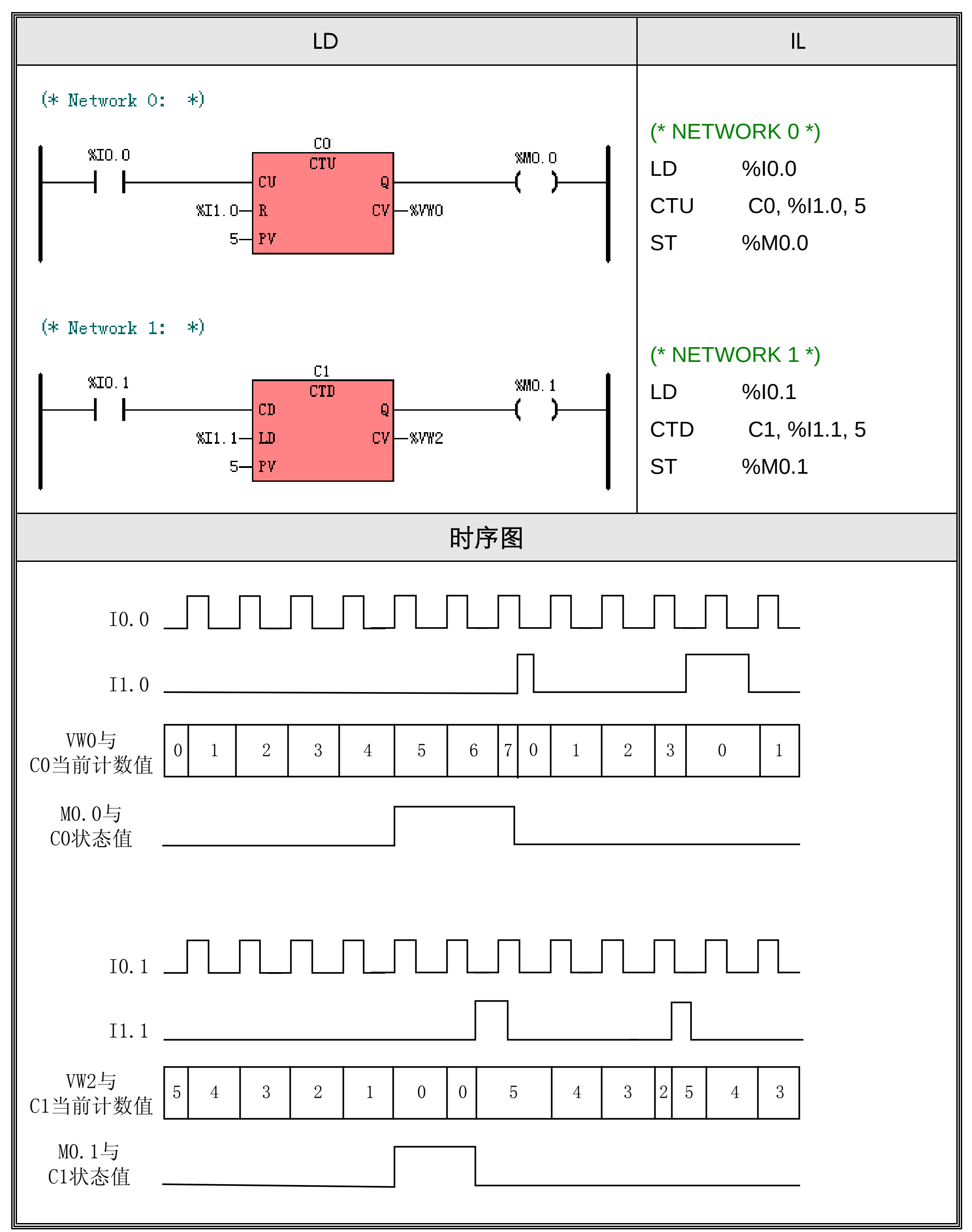
参数	输入/输出	数据类型	允许使用的内存区
Cx	-	计数器实例	C
CD	输入	BOOL	能量流
LD	输入	BOOL	I、Q、M、V、L、SM、T、C
PV	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
CV	输出	INT	Q、M、V、L、SM、AQ

在 LD 中，*CTD* 用于对 *CD* 输入端的上升沿从预设值 *PV* 进行递减（每次递减 1）计数，当实例 *Cx* 的当前计数值 *CV* 等于 0 时，其输出 *Q* 及其状态值均被置为 1。若 *LD* 输入端为 1，则实例 *Cx* 被复位，其输出 *Q* 及状态值均被置为 0，同时把预设值 *PV* 重新装入当前值 *CV*。

在 IL 中，*CTD* 用于对当前 CR 值的上升沿从预设值 *PV* 进行递减（每次递减 1）计数，当实例 *Cx* 的当前计数值等于 0 时，其状态值被置为 1。若 *LD* 输入端为 1，则实例 *Cx* 被复位，其状态值被置为 0，同时把预设值 *PV* 重新装入当前值。每次扫描 *CTD* 后，CR 值均被设置为实例 *Cx* 的状态值。

计数器实例 *Cx* 计数到 0 时将停止计数。

- CTU、CTD 使用举例


LD
IL
(* Network 0: *)
%I0.0
C0
CTU
CU
Q
%I1.0
R
CV
%VW0
5
PV
%M0.0
(* Network 1: *)
%I0.1
C1
CTD
CD
Q
%I1.1
LD
CV
%VW2
5
PV
%M0.1
(* NETWORK 0 *)
LD %I0.0
CTU C0, %I1.0, 5
ST %M0.0
(* NETWORK 1 *)
LD %I0.1
CTD C1, %I1.1, 5
ST %M0.1
时序图
I0.0
I1.0
VW0与
C0当前计数值
0 1 2 3 4 5 6 7 0 1 2 3 0 1
M0.0与
C0状态值
I0.1
I1.1
VW2与
C1当前计数值
5 4 3 2 1 0 0 5 4 3 2 5 4 3
M0.1与
C1状态值

6.12.2 高速计数器指令 --- HDEF、HSC

关于高速计数器的原理及其应用请参阅本手册第二部分中相关内容。

• 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	高速计数器定义	HDEF EN ENO HSC MODE		☑ CPU304 ☑ CPU306 ☑ CPU308
	高速计数器启动	HSC EN ENO N		
IL	高速计数器定义	HDEF HSC, MODE	U	
	高速计数器启动	HSC N		

参数	输入/输出	数据类型	描 述
HSC	输入	INT 型常量（0~5）	高速计数器编号
MODE	输入	INT 型常量（0~11）	为高速计数器 *HSC* 指定的工作模式
N	输入	INT 型常量（0~5）	高速计数器编号

HDEF 指令的作用是：为指定的高速计数器 *HSC* 分配一种工作模式 *MODE*。

HSC 指令的作用是：依据 SM 区中相应控制位、字节、双字的值来设置高速计数器 *N* 的工作模式并将其启动。

在 LD 中，*EN* 作为使能端决定了是否执行 *HDEF*、*HSC* 指令。

在 IL 中，由当前 CR 值决定了是否执行 *HDEF*、*HSC* 指令。*HDEF*、*HSC* 指令的执行不影响 CR 值。

- 高速计数器示例

LD

下面是一个简单的例子：利用 **HSC5** 来测量脉冲的频率（单位 **Hz**），脉冲由 **I0.3** 输入。

(* Network 0: SM0.4固定输出周期2s、占空比50%的脉冲。首先检测SM0.4的上升沿。 *)

%SM0.4 — R_TRIG (CLK, Q) — %M0.0 ( )

(* Network 1: 为HSC5设置新的计数值和预设值。 *)

%M0.0 — MOVE (EN, ENO; di#0 — IN, OUT — %SMD158) — MOVE (EN, ENO; di#9000 — IN, OUT — %SMD162) — (END)

(* Network 2: 利用SM0.4的上升沿设置HSC5的控制字节：增计数、启用新的计数值和预设值。 *)

%M0.0 — MOVE (EN, ENO; b#16#f8 — IN, OUT — %SMB157) — (END)

(* Network 3: 利用SM0.4的上升沿将HSC5的工作模式设置为0，并启动HSC5。 *)

%M0.0 — HDEF (EN, ENO; 5 — HSC; 0 — MODE) — HSC (EN, ENO; 5 — N) — (END)

(* Network 4: 在SM0.4的下降沿处取得HSC5的计数值，即可得到脉冲输入的频率。 *)

%SM0.4 — F_TRIG (CLK, Q) — MOVE (EN, ENO; %HC5 — IN, OUT — %VD20) — (END)

IL	下面是一个简单的例子：利用 **HSC5** 来测量脉冲的频率（单位 **Hz**），脉冲由 **I0.3** 输入。

```
(* NETWORK 0 *)
LD        %SM0.4              (* SM0.4 固定输出周期 2 秒、占空比 50%的脉冲 *)
R_TRIG                        (* 取得 SM0.4 的上升沿，在此上升沿处将开始计数 *)
MOVE   di#0, %SMD158          (* 将 HSC5 的计数值设置为 0 *)
MOVE   di#9000, %SMD162       (* 将 HSC5 的预设值设置为 9000 *)
MOVE   B#16#F8, %SMB157       (* 将 HSC5 设为增计数，并启用新的计数值和预设值 *)
HDEF    5, 0                  (* 将 HSC5 的工作模式设置为模式 0 *)
HSC     5                     (* 依据前面的设置启动 HSC5 *)

(* NETWORK 1 *)
LD          %SM0.4
F_TRIG                        (* 取得 SM0.4 的下降沿 *)
MOVE      %HC5, %VD20         (* 在此下降沿处取得 HSC5 的计数值 *)
                              (* 即可得到输入脉冲的频率 *)
```

6.12.3 PLS（高速脉冲输出）

关于高速脉冲输出的应用请参阅本手册第二部分中相关内容。

• 指令及其操作数说明

<table>
<tr><th></th><th>名称</th><th>指令格式</th><th>影响 CR 值</th><td rowspan="3">☑ CPU304
☑ CPU306
☑ CPU308</td></tr>
<tr><td>LD</td><td>高速脉冲输出</td><td>PLS
EN ENO
Q</td><td></td></tr>
<tr><td>IL</td><td>高速脉冲输出</td><td>PLS Q</td><td>U</td></tr>
</table>

参数	输入/输出	数据类型	描 述
Q	输入	INT 型常量（0 或 1）	指定高速输出通道： 0 表示在 Q0.0 输出； 1 表示在 Q0.1 输出。

PLS 指令的作用是：依据 SM 区中相应控制位、字节、双字的值来设置脉冲输出的特性并在 *Q* 指定的通道输出脉冲。

在 LD 中，*EN* 作为使能端决定了是否执行 *PLS* 指令。

在 IL 中，由当前 CR 值决定了是否执行 *PLS* 指令。*PLS* 指令的执行不影响 CR 值。

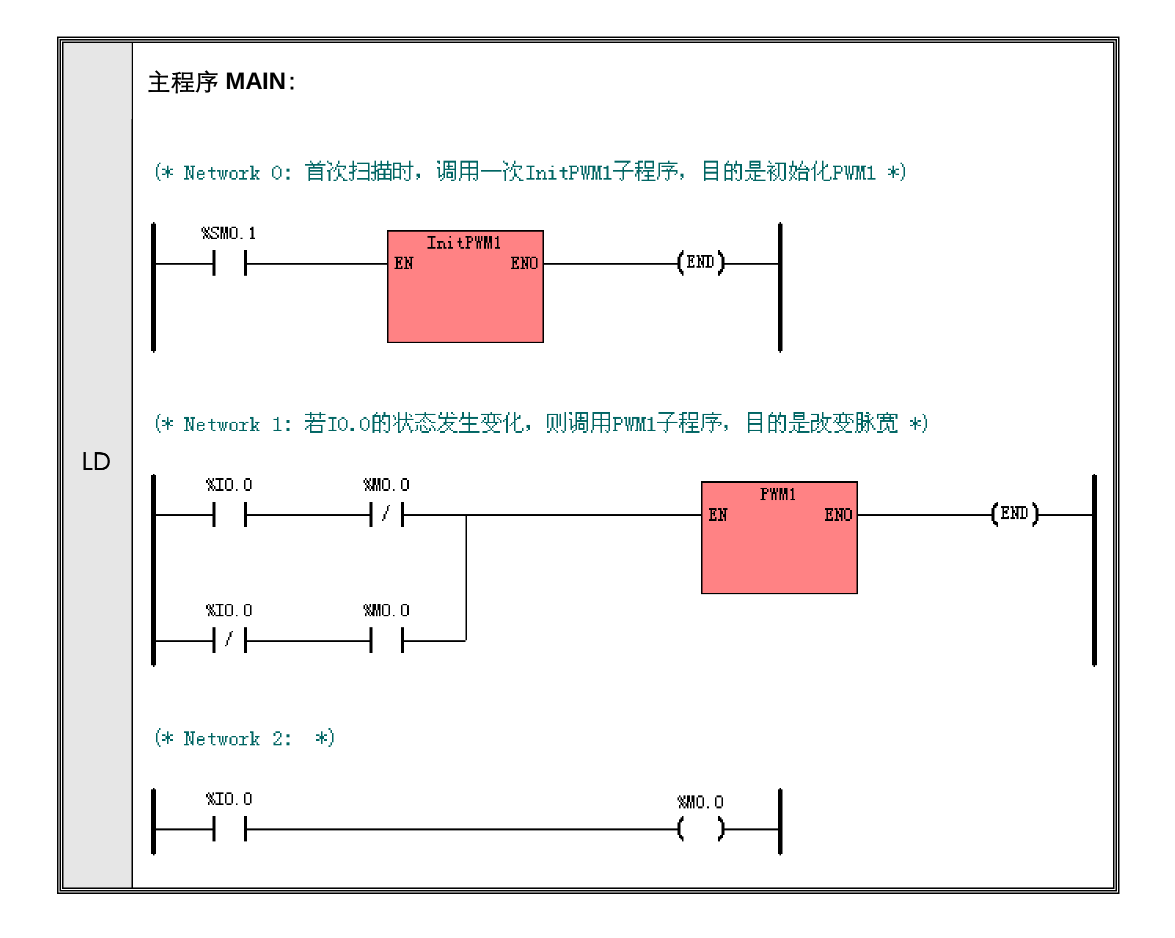
• PWM 示例

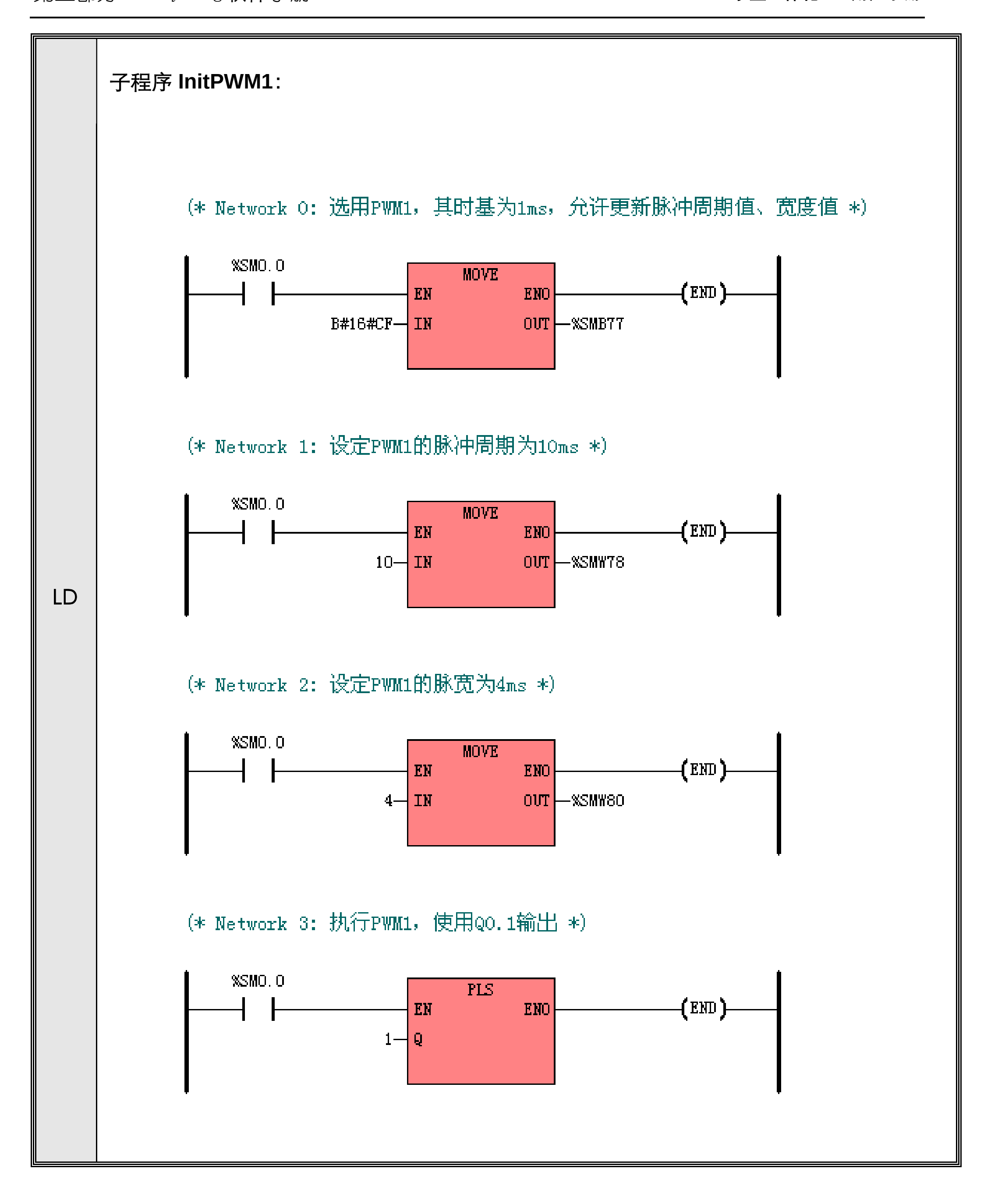
示例中使用了 PWM1，在 Q0.1 输出。

通过 I0.0 信号来改变输出脉冲的占空比，若 I0.0 为 0，则占空比为 40%；若 I0.0 为 1，则占空比为 80%。时序的示意图如下：

I0.0

Q0.0




LD
子程序 InitPWM1:
(* Network 0: 选用PWM1，其时基为1ms，允许更新脉冲周期值、宽度值 *)
%SM0.0
MOVE
EN
ENO
B#16#CF
IN
OUT
%SMB77
END
(* Network 1: 设定PWM1的脉冲周期为10ms *)
%SM0.0
MOVE
EN
ENO
10
IN
OUT
%SMW78
END
(* Network 2: 设定PWM1的脉宽为4ms *)
%SM0.0
MOVE
EN
ENO
4
IN
OUT
%SMW80
END
(* Network 3: 执行PWM1，使用Q0.1输出 *)
%SM0.0
PLS
EN
ENO
1
Q
END

LD	子程序 **PWM1**： (* Network 0: 若I0.0为0，则设定PWM1的脉宽为4ms *) %I0.0 —|/|— MOVE EN ENO —(END) 4— IN OUT —%SMW80 (* Network 1: 若I0.0为1，则设定PWM1的脉宽为8ms *) %I0.0 —| |— MOVE EN ENO —(END) 8— IN OUT —%SMW80 (* Network 2: 执行PWM1，使用Q0.1输出 *) %SM0.0 —| |— PLS EN ENO —(END) 1— Q

<table>
<tr><td rowspan="3">IL</td><td>

主程序 MAIN:

```
(* NETWORK 0 *)
LD      %SM0.1
CAL     InitPWM1        (* CPU 启动时调用一次 InitPWM1 子程序,初始化 PWM1 *)

LD      %I0.0
ANDN    %M0.0           (* 在 M0.0 中存放着上一次扫描时 I0.0 的值 *)
ST      %M0.1
LD      %M0.0
ANDN    %I0.0
OR      %M0.1
CAL     PWM1            (* 若 I0.0 的值发生变化，则调用 PWM1 子程序 *)

LD      %I0.0
ST      %M0.0           (* 将 I0.0 的状态存储于 M0.0 中 *)
```

</td></tr>
<tr><td>

子程序 InitPWM1:

```
(* NETWORK 0 *)
LD      %SM0.0
MOVE    B#16#CF, %SMB77 (* 选用 PWM1，设定其时基为 1ms 并允许其操作 *)
MOVE    10, %SMW78      (* 设定 PWM1 的脉冲周期为 10ms *)
MOVE    4, %SMW80       (* 设定 PWM1 的脉宽度为 4ms *)
PLS     1               (* 执行 PWM1，使用 Q0.1 输出 *)
```

</td></tr>
<tr><td>

子程序 PWM1:

```
(* NETWORK 0 *)
LDN     %I0.0           (* 若 I0.0 为 0 *)
MOVE    4, %SMW80       (* 则设定 PWM1 的脉宽为 4ms *)
LD      %I0.0           (* 若 I0.0 为 1 *)
MOVE    8, %SMW80       (* 则设定 PWM1 的脉宽为 8ms *)
PLS     1               (* 执行 PWM1，使用 Q0.1 输出 *)
```

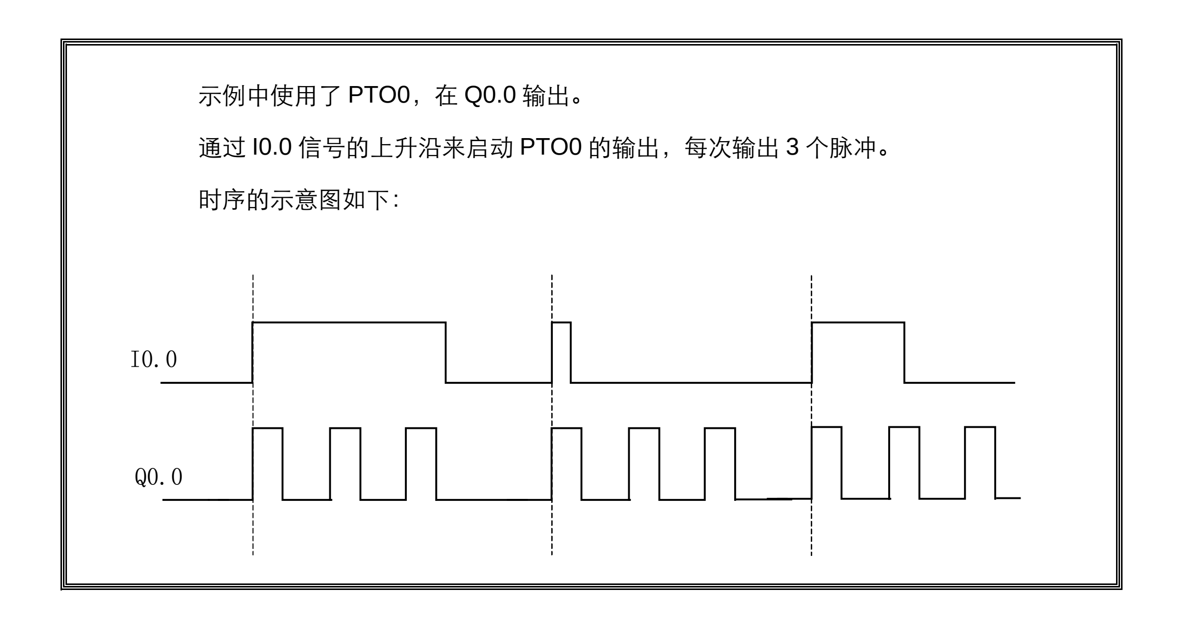
</td></tr>
</table>

• PTO 单段模式示例

示例中使用了 PTO0，在 Q0.0 输出。

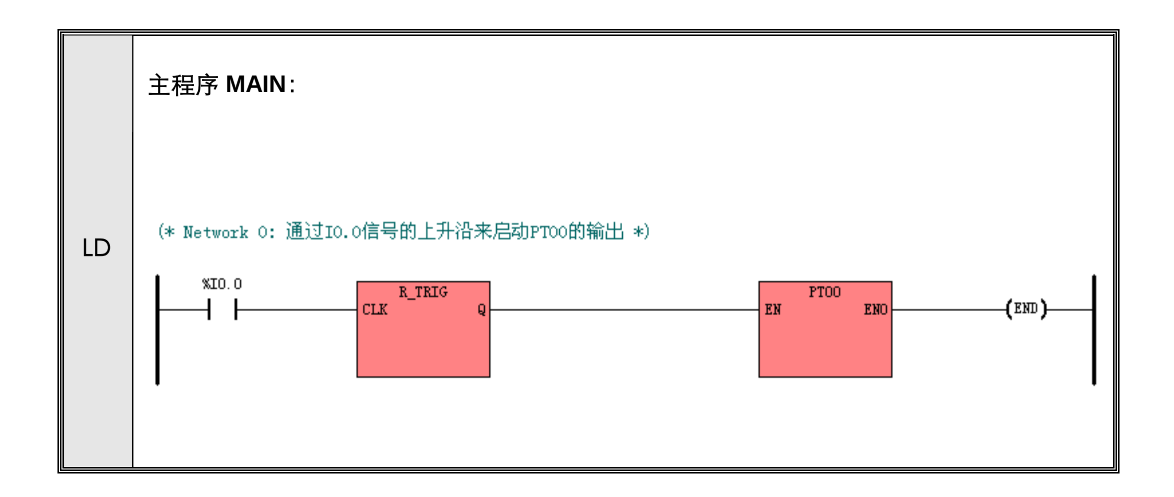
通过 I0.0 信号的上升沿来启动 PTO0 的输出，每次输出 3 个脉冲。

时序的示意图如下：



LD

主程序 MAIN：



LD

子程序 **PTO0**：

(* Network 0: 选用PTO0的单段模式输出。
设置时基为1ms，允许更新脉冲周期值、计数值 *)

%SM0.0 — MOVE (EN, ENO) — b#16#8F — IN, OUT — %SMB67 — (END)

(* Network 1: 设定输出脉冲周期为10ms *)

%SM0.0 — MOVE (EN, ENO) — 10 — IN, OUT — %SMW68 — (END)

(* Network 2: 设定每次输出3个脉冲 *)

%SM0.0 — MOVE (EN, ENO) — di#3 — IN, OUT — %SMD72 — (END)

(* Network 3: 执行PTO0，单段模式，在Q0.0输出 *)

%SM0.0 — PLS (EN, ENO) — 0 — Q — (END)

<table>
<tr><td rowspan="2">IL</td><td>

主程序 MAIN：

```
(* NETWORK 0 *)
LD      %I0.0
R_TRIG
CAL     PTO0                  (* 若检测到 I0.0 的上升沿，则启动 PTO0 输出 *)
```

</td></tr>
<tr><td>

子程序 PTO0：

```
(* NETWORK 0 *)
LD      %SM0.0
MOVE    b#16#8F, %SMB67       (* 选用 PTO0 的单段模式输出，时基 1ms 并允许操作 *)
MOVE    10, %SMW68            (* 设定输出脉冲周期为 10ms *)
MOVE    di#3, %SMD72          (* 设定每次输出 3 个脉冲 *)
PLS     0                     (* 执行 PTO0，单段模式，在 Q0.0 输出 *)
```

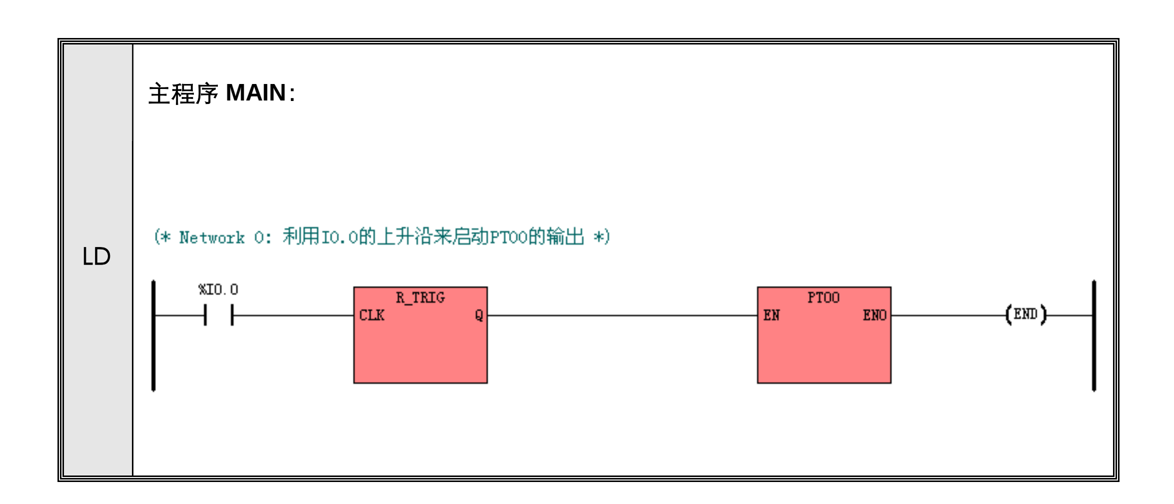
</td></tr>
</table>

• PTO 多段模式示例

示例中使用了 PTO0，在 Q0.0 输出。

通过 I0.0 信号的上升沿来启动 PTO0 的输出，要求输出如下图所示的波形来控制步进电机加速、匀速、减速运行。

频率
2.5kHz
500Hz
时间
第1段 400脉冲
第2段 600脉冲
第3段 400脉冲
总计1400 脉冲



LD

子程序 PTO0：

(* Network 0: 选用PTO0的多段模式，设置时基为1us，并允许其操作 *)

%SM0.0 — MOVE (EN/ENO) IN: b#16#A0 — OUT: %SMB67 — (END)

(* Network 1: 设置包络表的起始位置为VB1 *)

%SM0.0 — MOVE (EN/ENO) IN: 1 — OUT: %SMW168 — (END)

(* Network 2: 设置总段数为3段 *)

%SM0.0 — MOVE (EN/ENO) IN: b#3 — OUT: %VB1 — (END)

(* Network 3: 设置第1段的起始周期为2000us，周期增量为-4（即减速运行） *)

%SM0.0 — MOVE (EN/ENO) IN: 2000 — OUT: %VW2 — MOVE (EN/ENO) IN: -4 — OUT: %VW4 — (END)

(* Network 4: 设置第1段输出400个脉冲 *)

%SM0.0 — MOVE (EN/ENO) IN: di#400 — OUT: %VD6 — (END)

LD

子程序 PTO0：（接上页）

(* Network 5：设置第2段的起始周期为400us，周期增量为0（即匀速运行） *)

%SM0.0 — MOVE (EN, ENO; 400—IN, OUT—%VW10) — MOVE (EN, ENO; 0—IN, OUT—%VW12) — (END)

(* Network 6：设置第2段输出400个脉冲 *)

%SM0.0 — MOVE (EN, ENO; di#600—IN, OUT—%VD14) — (END)

(* Network 7：设置第3段的起始周期为400us，周期增量为4（即加速运行） *)

%SM0.0 — MOVE (EN, ENO; 400—IN, OUT—%VW18) — MOVE (EN, ENO; 4—IN, OUT—%VW20) — (END)

(* Network 8：设置第3段输出400个脉冲 *)

%SM0.0 — MOVE (EN, ENO; di#400—IN, OUT—%VD22) — (END)

(* Network 9：执行PTO0，多段模式，在Q0.0输出 *)

%SM0.0 — PLS (EN, ENO; 0—Q) — (END)

IL

主程序 MAIN：

```
(* NETWORK 0 *)
LD      %I0.0
R_TRIG
CAL     PTO0              (* 若检测到 I0.0 的上升沿，则启动 PTO0 输出 *)
```

子程序 PTO0：

```
(* NETWORK 0 *)
LD      %SM0.0
MOVE    b#16#A0, %SMB67   (* 选用 PTO0 的多段模式，时基为 1us，并允许其操作 *)
MOVE    1, %SMW168        (* 设置包络表的起始位置为 VB1 *)
MOVE    b#16#03, %VB1     (* 总段数：3 段 *)

(* 第 1 段包络 *)
MOVE    2000, %VW2        (* 设置第 1 段的起始周期为 2000us *)
MOVE    -4, %VW4          (* 设置第 1 段的周期增量为-4（即减速运行）*)
MOVE    di#400, %VD6      (* 设置第 1 段输出 400 个脉冲 *)

(* 第 2 段包络 *)
MOVE    400, %VW10        (* 设置第 1 段的起始周期为 400us *)
MOVE    0, %VW12          (* 设置第 2 段的周期增量为 0（即匀速运行）*)
MOVE    di#600, %VD14     (* 设置第 2 段输出 600 个脉冲 *)

(* 第 3 段包络 *)
MOVE    400, %VW18        (* 设置第 3 段的起始周期为 400us *)
MOVE    4, %VW20          (* 设置第 3 段的周期增量为 4（即减速运行） *)
MOVE    di#400, %VD22     (* 设置第 3 段输出 400 个脉冲 *)

PLS     0                 (* 执行 PTO0，多段模式，在 Q0.0 输出*)
```

6.13 定时器

定时器是 IEC61131-3 标准中定义的功能块之一，共有 TON、TOF 和 TP 三种。关于功能块及其实例的使用请参阅 3.6.4 FB 实例存储区的分配 一节。

6.13.1 TON（接通延时定时器）

• TON 及其操作数说明

	名称	指令格式	影响 CR 值	
LD	接通延时定时器	Tx TON IN Q PT ET		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	接通延时定时器	TON Tx, PT	P	

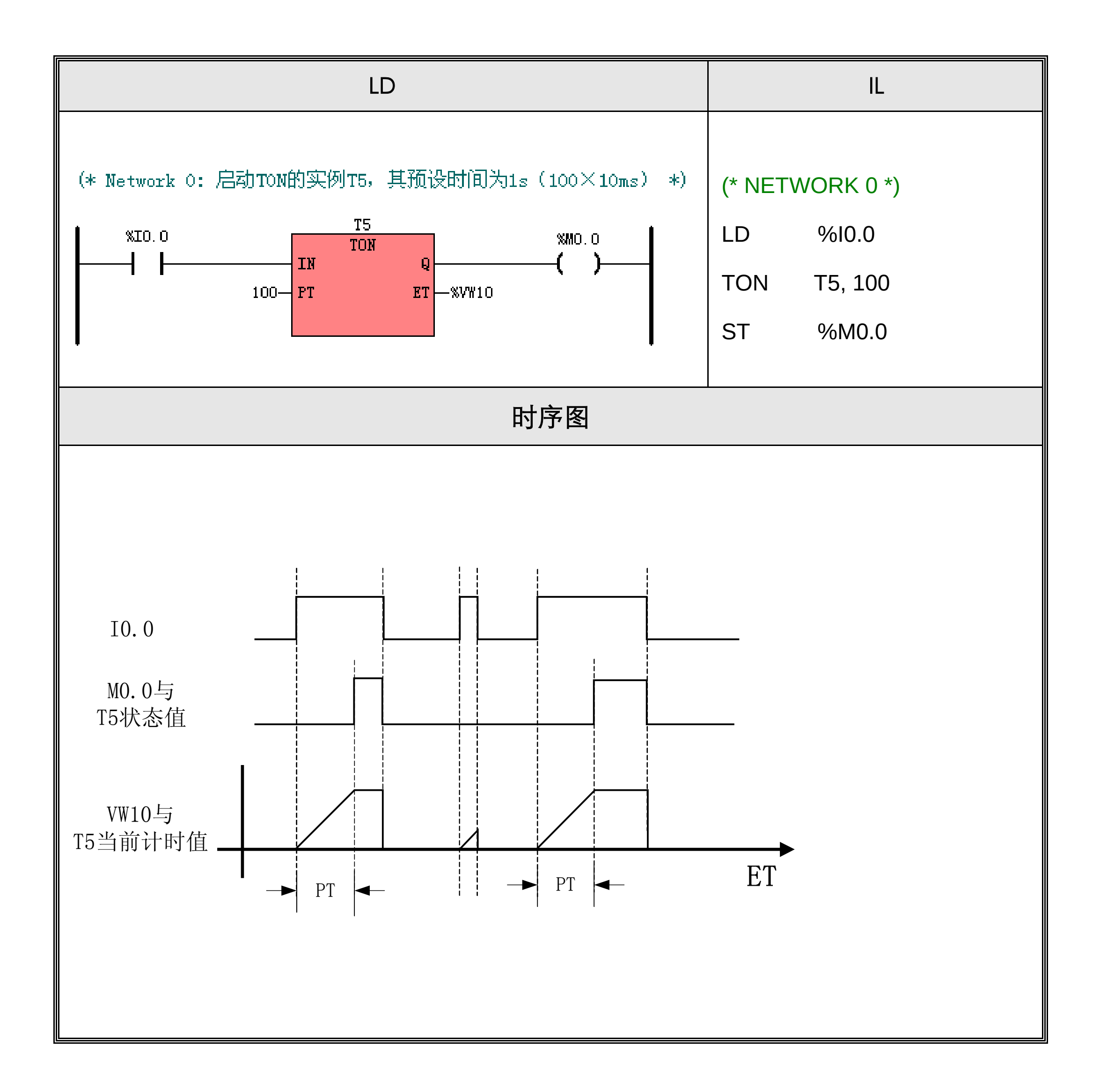
参数	输入/输出	数据类型	允许使用的内存区
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

参数 *PT* 代表的预设时间为：实例 *Tx* 的时基乘以 *PT* 值。

在 LD 中，若在输入端 *IN* 检测到上升沿，则实例 *Tx* 开始启动定时，当 *Tx* 的当前计时值 *ET* 大于等于预设时间 *PT* 时，*Tx* 停止，其输出 *Q* 及其状态值均被置为 1。若输入 *IN* 变为 0，则 *Tx* 被复位，其输出 *Q* 及状态值均被置为 0，同时 *ET* 及其当前计时值也被清零。

在 IL 中，若检测到当前 CR 值的上升沿，则实例 *Tx* 开始启动定时，当 *Tx* 的当前计时值大于等于预设时间 *PT* 时，*Tx* 停止，其状态值被置为 1。若当前 CR 值变为 0，则 *Tx* 被复位，其状态值及当前计时值均被清零。每次扫描 *TON* 后，CR 值均被设置为实例 *Tx* 的状态值。

- TON 使用举例



6.13.2　TOF（断开延时定时器）

- TOF 及其操作数说明

	名称	指令格式	影响 CR 值	
LD	断开延时定时器	Tx TOF IN　Q PT　ET		☑ CPU304 ☑ CPU306 ☑ CPU308
IL	断开延时定时器	TOF　Tx, PT	P	

参数	输入/输出	数据类型	允许使用的内存区
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

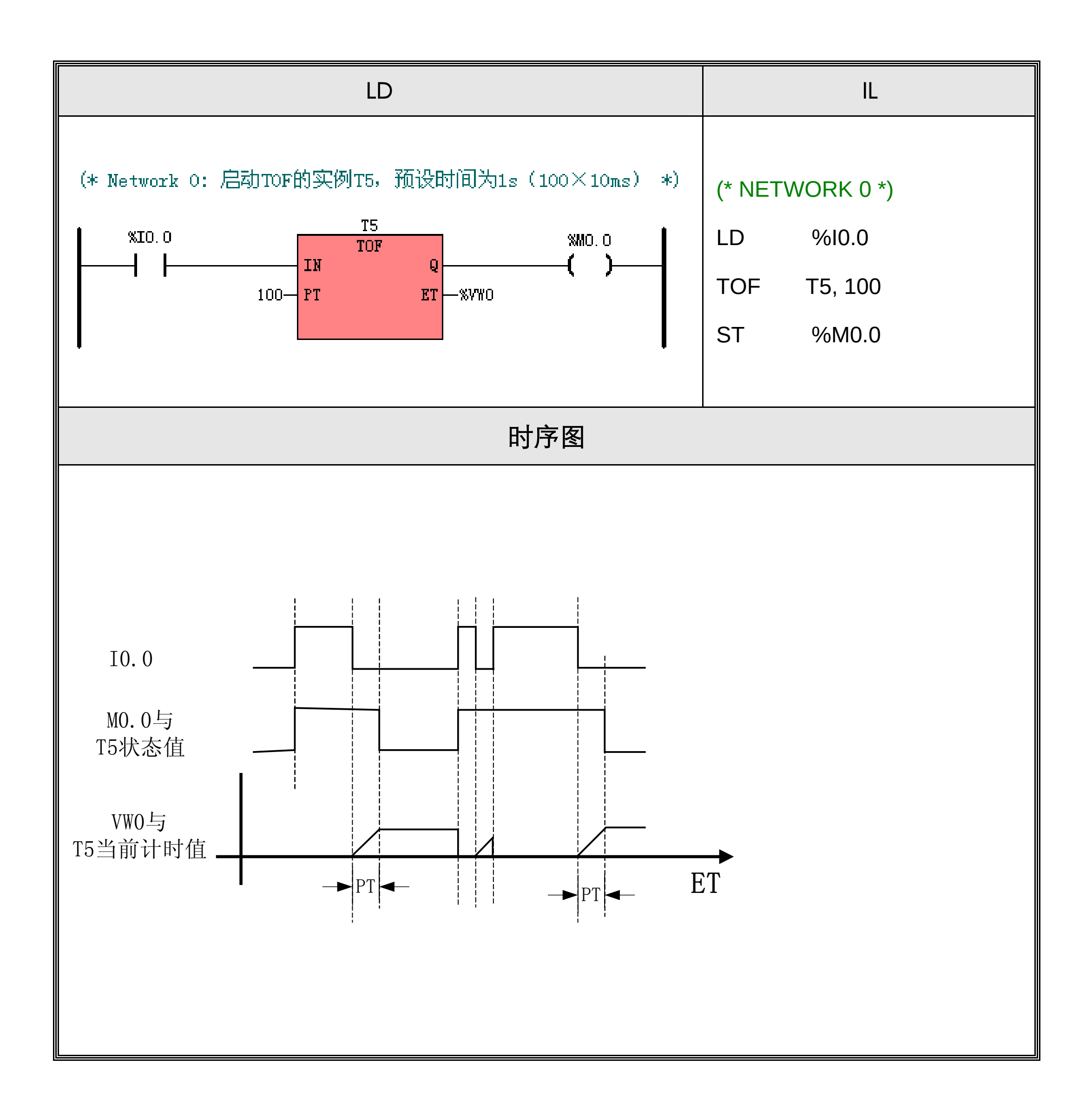
参数 *PT* 代表的预设时间为：实例 *Tx* 的时基乘以 *PT* 值。

在 LD 中，若在输入端 *IN* 检测到下降沿，则实例 *Tx* 开始启动定时，当 *Tx* 的当前计时值 *ET* 大于等于预设时间 *PT* 时，*Tx* 停止，其输出 *Q* 及其状态值均被置为 0。若输入 *IN* 变为 1，则 *Tx* 被复位，其输出 *Q* 及状态值均被置为 1，同时 *ET* 及其当前计时值被清零。若输入 IN 产生了下降沿且保持为 0 的时间小于预设时间，则 *Tx* 的输出 *Q* 及其状态值一直保持为 1。

在 IL 中，若检测到当前 CR 值的下降沿，则实例 *Tx* 开始启动定时，当 *Tx* 的当前计时值大于等于预设时间 *PT* 时，*Tx* 停止，其状态值被置为 0。若当前 CR 值变为 1，则 *Tx* 被复位，其状态值被置为 1，且当前计时值被清零。若当前 CR 值产生了下降沿且保持为 0 的时间小于预设时间，

则 *Tx* 的状态值一直保持为 1。每次扫描 *TOF* 后，CR 值均被设置为实例 *Tx* 的状态值。

- TOF 使用举例



6.13.3　TP（脉冲定时器）

- TP 及其操作数说明

<table>
<tr><th></th><th>名称</th><th>指令格式</th><th>影响 CR 值</th><td rowspan="3">☑ CPU304
☑ CPU306
☑ CPU308</td></tr>
<tr><td>LD</td><td>脉冲定时器</td><td>Tx
TP
IN　Q
PT　ET</td><td></td></tr>
<tr><td>IL</td><td>脉冲定时器</td><td>TP　Tx, PT</td><td>P</td></tr>
</table>

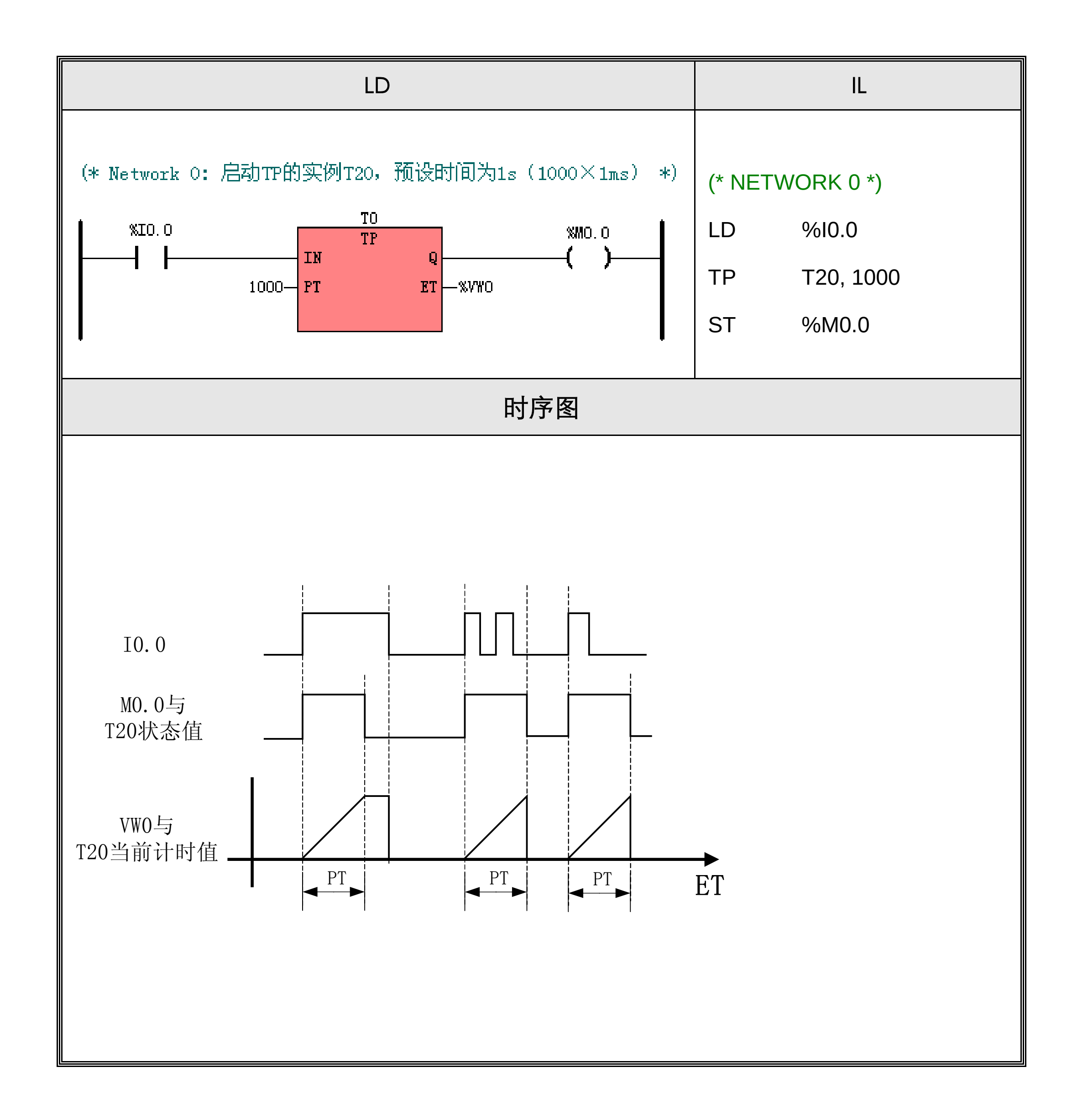
参数	输入/输出	数据类型	允许使用的内存区
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

参数 *PT* 代表的预设时间为：实例 *Tx* 的时基乘以 *PT* 值。

在 LD 中，若在输入端 *IN* 检测到上升沿，则实例 *Tx* 开始启动定时，在其输出端 *Q* 及其状态值输出一个恒定宽度的脉冲，脉宽值为预设时间 *PT*。参数 *ET* 中存放着 *Tx* 的当前计时值。

在 IL 中，若检测到当前 CR 值的上升沿，则实例 *Tx* 开始启动定时，在其状态值输出一个恒定宽度的脉冲，脉宽值为预设时间 *PT*。每次扫描 *TP* 后，CR 值均被设置为实例 *Tx* 的状态值。

• TP 使用举例


LD
IL
(* Network 0: 启动TP的实例T20，预设时间为1s（1000×1ms） *)
%I0.0
T0
TP
IN
Q
1000
PT
ET
%VW0
%M0.0
(* NETWORK 0 *)
LD %I0.0
TP T20, 1000
ST %M0.0
时序图
I0.0
M0.0与
T20状态值
VW0与
T20当前计时值
PT
PT
PT
ET

附录 A　使用 Modbus RTU 协议与第三方设备通讯

此处的“第三方设备”表示不是由凯迪恩公司提供的设备。

在默认的情况下，KDN-K3 系列 PLC 作为 Modbus 从站。

1、 PLC 内存区说明

1.1 Modbus 可访问的内存区

利用 Modbus 协议可访问的 K3 系列 PLC 内存区域分类如下：

开关量输出：DO、M 区

开关量输入：DI、M 区

模拟量输入：AI、V 区

模拟量输出：AQ、V 区

1.2 CPU306 的内存区域对照

区域名称	PLC 地址范围	类型	对应的 Modbus 寄存器范围
I	I0.0 – I7.7	开关量输入	0 – 63(1)
Q	Q0.0 – Q7.7	开关量输出	0 – 63(1)
M	M0.0 – M31.7	开关量输入/输出	64 – 319(1)
AI	AIW0 – AIW30	模拟量输入	0 – 15(1)
AQ	AQW0 – AQW30	模拟量输出	0 – 15(1)
V	VW0 – VW4094	模拟量输入/输出	16 – 2063(1)

（1） 某些设备中的 Modbus 寄存器是从 1 开始定义的，此时将表中的数据直接加 1 即可。

2、 Modbus RTU 报文基本格式

起始应有小于 3.5 个字符的报文间隔	目标站号	功能码	数据	CRC 校验码
	1 字节	1 字节	N 字节	2 字节

2.1 Modbus 命令简介

注：下面对于各请求命令的“应答格式“的描述是指命令被正确执行时的应答格式。若 CPU 接收到错误的命令或者命令被执行错误，则返回的应答帧中“功能码”部分变为如下数据：功能码的最高位置 1 后得到的数据。比如功能码为 01，若响应错误，则返回的功能码为 0x81。

2.1.1 功能码 01：读线圈（开关量输出）

请求格式：

目标站号	功能码	起始地址高字节	起始地址低字节	读取个数高字节	读取个数低字节	CRC
1 字节	01	1 字节	1 字节	1 字节	1 字节	2 字节

正确应答格式：

站号	功能码	返回数据字节数	返回数据字节 1	返回数据字节 2	…	CRC
1 字节	01	1 字节	1 字节	1 字节	…	2 字节

2.1.2 功能码 02：读输入状态（开关量输入）

请求格式：

目标站号	功能码	起始地址高字节	起始地址低字节	读取个数高字节	读取个数低字节	CRC
1 字节	02	1 字节	1 字节	1 字节	1 字节	2 字节

正确应答格式:

站号	功能码	返回数据字节数	返回数据字节 1	返回数据字节 2	…	CRC
1 字节	02	1 字节	1 字节	1 字节	…	2 字节

2.1.3 功能码 03：读保持寄存器（模拟量输出）

请求格式:

目标站号	功能码	起始地址高字节	起始地址低字节	读取个数高字节	读取个数低字节	CRC
1 字节	03	1 字节	1 字节	1 字节	1 字节	2 字节

正确应答格式:

站号	功能码	返回数据字节数	返回数据字节 1	返回数据字节 2	…	CRC
1 字节	03	1 字节	1 字节	1 字节	…	2 字节

2.1.4 功能码 04：读输入寄存器（模拟量输入）

请求格式:

目标站号	功能码	起始地址高字节	起始地址低字节	读取个数高字节	读取个数低字节	CRC
1 字节	04	1 字节	1 字节	1 字节	1 字节	2 字节

正确应答格式:

站号	功能码	返回数据字节数	返回数据字节 1	返回数据字节 2	…	CRC
1 字节	04	1 字节	1 字节	1 字节	…	2 字节

2.1.5 功能码 05：写单线圈（开关量输出）

请求格式：

目标站号	功能码	线圈地址 高字节	线圈地址 低字节	强制值 高字节	强制值 低字节	CRC 校验码
1 字节	05	1 字节	1 字节	1 字节	1 字节	2 字节

注：强制值 = 0xFF00，则置线圈为 ON；强制值＝0x0000，则置线圈为 OFF。

应答格式：

若设置成功，原文返回

2.1.6 功能码 06：写单保持寄存器（模拟量输出）

请求格式：

目标站号	功能码	寄存器地址 高字节	寄存器地址 低字节	强制值 高字节	强制值 低字节	CRC 校验码
1 字节	05	1 字节	1 字节	1 字节	1 字节	2 字节

应答格式：

若设置成功，原文返回

2.1.7 功能码 15：写多线圈（开关量输出）

请求格式：

目标 站号	功 能 码	起始地址 高字节	起 始 地 址 低字节	数量 高字节	数量 低字节	强 制 值 字节数	强制值 第 1 字节	…	CRC
1 字节	15	1 字节	1 字节	1 字节	1 字节	1 字节	1 字节	…	2 字节

正确应答格式：

目标站号	功能码	起始地址高字节	起始地址低字节	数量高字节	数量低字节	CRC 校验码
1 字节	15	1 字节	1 字节	1 字节	1 字节	2 字节

2.1.8 功能码 16：写多保持寄存器（模拟量输出）

请求格式：

目标站号	功能码	起始地址高字节	起始地址低字节	数量高字节	数量低字节	强制值字节数	强制值 1 高字节	强制值 1 低字节	…	CRC
1 字节	15	1 字节	1 字节	1 字节	1 字节	1 字节	1 字节	1 字节	…	2 字节

正确应答格式：

目标站号	功能码	起始地址高字节	起始地址低字节	数量高字节	数量低字节	CRC 校验码
1 字节	15	1 字节	1 字节	1 字节	1 字节	2 字节

2.2 Modbus 协议中的 CRC 校验算法

在 Modbus RTU 协议中，使用 CRC 作为帧的校验方式。

下面是用 C 编写的两种 CRC 算法：

2.2.1 直接计算 CRC

```
/* 参  数：chData —— const BYTE*，指向待校验数据存储区的首地址
   uNO  —— 待校验数据的字节个数
   返回值：WORD 型，计算出的 CRC 值。                */
WORD   CalcCrc(const BYTE* chData, WORD uNo)
{
      WORD crc=0xFFFF;
       WORD wCrc;
       UCHAR i,j;
```

```
    for (i=0; i<uNo; i++)
    {
        crc ^= chData[i];
        for (j=0; j<8; j++)
        {
            if (crc & 1)
            {
                crc >>= 1;
                crc ^= 0xA001;
            }
            else
                crc >>= 1;
        }
    }

    wCrc=( (WORD)LOBYTE(crc) )<<8;
    wCrc=wCrc|( (WORD)HIBYTE(crc) );
    return (wCrc);
}
```

2.2.2 查表快速计算 CRC

```
/* 高字节 CRC 表 */
const UCHAR auchCRCHi[] =
{
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
        0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
        0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
        0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
        0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
```

```
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
        0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
        0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
        0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
        0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
        0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
        0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
        0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
        0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
} ;

/* 低字节 CRC 表 */
const UCHAR auchCRCLo[] =
{
        0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
        0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
        0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
        0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
        0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
        0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
        0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
        0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
        0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
        0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
        0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
        0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
```

```
        0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
        0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
        0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
        0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
        0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
        0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
        0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
        0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
        0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
        0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
        0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
        0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
        0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
        0x43, 0x83, 0x41, 0x81, 0x80, 0x40
} ;

/* 参  数：puchMsg  —— const BYTE*，指向待校验数据存储区的首地址
   usDataLen —— 待校验数据的字节个数
   返回值：WORD 型，计算出的 CRC 值。    */
WORD CKDNSerialCom::CalCrcFast(const BYTE* puchMsg , WORD usDataLen)
{
      BYTE uchCRCHi = 0xFF ;   /* CRC 高字节初始化  */
      BYTE uchCRCLo = 0xFF ;   /* CRC 低字节初始化  */
       WORD uIndex ;            /* CRC 查表的索引*/

       while (usDataLen--)
       {
            uIndex = uchCRCHi ^ *puchMsg++ ;        /*  计算  CRC */
            uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex];
            uchCRCLo = auchCRCLo[uIndex] ;
       }
       return (uchCRCHi << 8 | uchCRCLo) ;
}
```

附录 B 系统存储器 SM 区的定义

本附录详细描述了系统存储区 SM 区的定义。系统存储器是用来辅助 KDN-K3 PLC 实现一定的指令功能，用户也可以通过使用系统存储器来控制 PLC 的某些状态。

1、 SMB0

SM0.0 - SM0.7 位由 CPU 系统软件进行赋值，不受用户程序控制，在用户程序中只能对这些位调用（只读）。每一位功能描述见表 B-1。

SM 位(只读)	说明
SM0.0	总是为”1”
SM0.1	CPU 首次扫描时为”1”，之后清”0”。通常用于用户程序初始化。
SM0.2	如果 RAM 内数据丢失，首次扫描循环时为”1”，之后清”0”。
SM0.3	频率为 1Hz 的脉冲，占空比为 50%。
SM0.4	频率为 2Hz 的脉冲，占空比为 50%。
SM0.5	频率为 4Hz 的脉冲，占空比为 50%。
SM0.6	频率为 60Hz 的脉冲，占空比为 50%。
SM0.7	当 CPU 进入运行模式时，从首次扫描循环开始时，在用户程序设定的时间内保持为”1”，此后为”0”。可用于提供机器预热时间。

2、SMW26 和 SMW28

SMW26 和 SMW28 用于存储顶调电位器的调节值。存储过程由 CPU 系统软件自动完成，无需用户程序干预。SMW26 和 SMW28 对用户来说是只读的。

SMW26 对应顶调电位器 0 的数值，SMW28 对应顶调电位器 1 的数值。

3、SMB31 和 SMW32

SMB31 和 SMW32 用于控制将 V 区中指定的的值写入 FRAM（铁电存储器）中，以便在 PLC 长期断电的情况下保持数据。

写 FRAM 操作的方式由 SMB31 的值来确定。注意：如果在写 FRAM 命令被执行之前，SMB31 被多次赋值，则按最近的一次赋值进行操作。

3.1 SM31.0、SM31.1 以及 SM31.7

SM31.0	SM31.1	描 述
0	0	写入一个字节
0	1	写入一个字节
1	0	写入一个字（2 字节）
1	1	写入一个双字（4 字节）

SM31.7	描述
1	允许写 FRAM。
0	不允许写 FRAM。

3.2 SMW32

在 SMW32 中存放着要 V 区中要写入 FRAM 的连续数据的起始字节地址。

允许写入 FRAM 的数据为：VB3648~VB3902，共 255 个字节。

3.3 写 FRAM 的命令

写 FRAM 的命令为： **MOVE offset，%SMW32**

参数 *offset* 为 INT 型常量，代表 V 区中将要写入 FRAM 的连续数据的起始字节地址。比如，

要将 VB3600 写入 FRAM，则将此参数赋值为 3600。

注意：在 CPU 每次扫描的最后，进行实际的写 FRAM 操作，所以，将要保存的内存地址在 CPU 一次扫描最后时刻的值才会被写入 FRAM。用户在实际编程时，必须考虑并遵守此规则，才能达到预期的效果。下面将以 IL 编写一个实力示例。

```
(* 由 M0.0 控制将%VB3649、%VW3650、%VD3652 写入到 FRAM *)

(*NETWORK   0*)
LDN     %M0.0               (* 若 M0.0 为 0 *)
MOVE    B#0, SMB31          (* 则不允许写 FRAM *)

(*NETWORK   1*)
LD      %M0.0                     (* 若 M0.0 为 1   *)
MOVE    B#2#10000001, SMB31       (* 表示写入一个字节  *)
MOVE    3649，%SMW32              (* 写入 VB3649 *)
MOVE    B#2#10000010, SMB31       (* 表示写入一个字（2 字节）*)
MOVE    3650，%SMW32              (* 写入%VW3650 *)
MOVE    B#2#10000011, SMB31       (* 表示写入一个双字（4 字节）*)
MOVE    3652，%SMW32              (* 写入%VD3652 *)
```

4、SMB36 到 SMB66

SMB36到SMB65用于监测和控制高速计数器 HSC 0 、HSC 1和HSC 2的操作，见下表。

SM	描　　述
SM36.0 ~SM36.4	保留
SM36.5	HSC0 当前计数方向位：1 – 增计数
SM36.6	HSC0 当前值等于预设值位：1 – 等于
SM36.7	HSC0 当前值大于预设值位：1 – 大于

SM37.0	HSC0 复位有效电平控制位：0 – 高电平，1 – 低电平
SM37.1	HSC0 复位有效电平控制位：0 – 高电平，1 – 低电平
SM37.2	HSC0 正交计数器速率选择：0 –4×速率，1 – 1×速率
SM37.3	HSC0 方向控制位：1 – 增计数
SM37.4	HSC0 更新方向：1 – 更新方向
SM37.5	HSC0 更新预设值：1 – 向 HSC0 写新的预设值
SM37.6	HSC0 更新当前值：1 – 向 HSC0 写新的当前值
SM37.7	HSC0 有效位：1 – 有效
SMD38	HSC0 新的当前值。
SMD42	HSC0 新的预设值。
SM46.0~SM46.4	保留
SM46.5	HSC1 当前计数方向位：1 – 增计数
SM46.6	HSC1 当前值等于预设值位：1 – 等于
SM46.7	HSC1 当前值大于预设值位：1 – 大于
SM47.0	HSC1 复位有效电平控制位：0 –高电平，1 – 低电平
SM47.1	HSC1 复位有效电平控制位：0 – 高电平，1 – 低电平
SM47.2	HSC1 正交计数器速率选择：0 –4×速率，1 – 1×速率
SM47.3	HSC1 方向控制位：1 – 增计数
SM47.4	HSC1 更新方向：1 – 更新方向
SM47.5	HSC1 更新预设值：1 – 向 HSC0 写新预设值
SM47.6	HSC1 更新当前值：1 – 向 HSC0 写新当前值
SM47.7	HSC1 有效位：1 – 有效
SMD48	HSC1 新的当前值。
SMD52	HSC1 新的预设值。
SM56.0~SM56.4	保留
SM56.5	HSC2 当前计数方向位：1 – 增计数
SM56.6	HSC2 当前值等于预设值位：1 – 等于
SM56.7	HSC2 当前值大于预设值位：1 – 大于
SM57.0	HSC2复位有效电平控制位：0 – 高电平，1 – 低电平
SM57.1	HSC2复位有效电平控制位：0 – 高电平，1 – 低电平
SM57.2	HSC2正交计数器速率选择：0 –4×速率，1 – 1×速率
SM57.3	HSC2方向控制位：1 – 增计数

SM57.4	HSC2更新方向：1 – 更新方向
SM57.5	HSC2更新预设值：1 – 向HSC0写新的预设值
SM57.6	HSC2更新当前值：1 – 向HSC0写新的当前值
SM57.7	HSC2有效位：1 – 有效
SMD58	HSC2新的当前值。
SMD62	HSC2新的预设值。

5、SMB66 到 SMB85

SMB66到SMB85用于监测和控制脉冲输出（PTO）和脉宽调制（PWM）功能。见下表。

SM	描　　述
SM66.0~SM66.3	保留
SM66.4	PTO 0 包络溢出
SM66.5	PTO 0 包络溢出
SM66.6	保留
SM66.7	PTO 0 空闲位：0 – PTO 忙， 1 – PTO 0 空闲
SM67.0	PTO 0 / PWM 0 更新周期： 1 – 写新的周期值
SM67.1	PWM 0 更新脉冲宽度值： 1 – 写新的脉冲宽度值
SM67.2	PTO 0 更新脉冲量： 1 – 写新的脉冲量
SM67.3	PTO 0 / PWM 0 基准时间单元：0 – 1us ， 1 – 1ms
SM67.4	保留
SM67.5	PTO 0 操作： 0 – 单段操作；1 – 多段操作。
SM67.6	PTO 0 / PWM 0 模式选择： 0 – PTO ， 1 – PWM
SM67.7	PTO 0 / PWM 0 有效位： 1 – 有效
SMW68	PTO 0 / PWM 0 周期（2 ~ 65535×时间基准）。
SMW70	PWM 0周期（0 ~ 65535×时间基准）。
SMD72	PTO 0 脉冲计数值（$1 \sim 2^{32}-1$）
SM76.0~SM76.3	保留
SM76.4	PTO 1包络溢出
SM76.5	PTO 1包络溢出
SM76.6	保留

SM76.7	PTO 1　空闲位：0 – PTO 忙， 1 – PTO 0　空闲
SM77.0	PTO 1 / PWM 1更新周期：1 – 写新周期值
SM77.1	PWM 1 更新脉冲宽度值：1 – 写新脉冲宽度值
SM77.2	PTO 1 更新脉冲量：1 – 写新脉冲量
SM77.3	PTO 1 / PWM 1基准时间单元：0 – 1us ， 1 – 1ms
SM77.4	保留
SM77.5	PTO 1 操作：0 – 单段操作；1 – 多段操作。
SM77.6	PTO 1 / PWM 1 模式选择：0 – PTO ， 1 – PWM
SM77.7	PTO 1 / PWM 1 有效位：1 – 有效
SMW78	PTO 1 / PWM 1 周期（2 ~ 65535×时间基准）
SMW80	PWM 1 周期（0 ~ 65535×时间基准）
SMD82	PTO 1脉冲计数值（1 ~ 2^{32} -1）

6、SMB86 到 SMB94

6.1 SMB86

用于存放 RCV 指令的状态，见下表

位（只读）		值	含 义
PORT 0	PORT 1		
SM86.0		1	接收到的字符存在奇偶校验错误，但不停止接收。
SM86.1		1	终止接收：达到最大接收字节数（见 SMB94）。
SM86.2		1	终止接收：接收一个字符超时（见 SMW92）。
SM86.3		1	终止接收：系统接收超时。
SM86.4		-	保留。
SM86.5		1	终止接收：接收到了用户定义的结束字符（见 SMB89）。
SM86.6		1	终止接收：参数设置错误，无起始条件接收或者无接收结束条件等。
SM86.7		1	终止接收：用户使用了禁止接收命令（见 SM87.7 ）

6.2 SMB87

位		值	描 述
PORT 0	PORT 1		
SM87.0		-	保留。
SM87.1		0	当发送或者接收完成时禁止生成相应的中断。
		1	当发送或者接收完成时允许生成相应的中断。
SM87.2		0	忽略 SMW92 中用户定义的接收字符超时值。
		1	使用 SMW92 中用户定义的接收字符超时值。
SM87.3		-	保留。
SM87.4		0	忽略 SMW90 中用户定义的接收准备时间。
		1	使用 SMW90 中用户定义的接收准备时间来作为启动接收的条件。
SM87.5		0	忽略 SMB89 中用户定义的接收结束字符。
		1	使用 SMB89 中用户定义的接收结束字符。
SM87.6		0	忽略 SMB88 中用户定义的接收开始字符。
		1	使用 SMB88 中用户定义的接收开始字符。
SM87.7		0	禁止接收数据。此禁止条件优先于其它所有的接收控制字。
		1	允许接收数据。

6.3 SMB88~SMW94

控制字（字节）		描述
PORT 0	PORT 1	
SMB88		用于存放用户定义的接收起始字符。 执行 RCV 指令后，CPU 收到了起始字符就开始进入有效接收状态，之前收到的数据都会被丢弃。CPU 将起始字符作为接收的第一个有效数据。 如果要使本设置值生效，需要将 SM87.6 置为 1。

SMB89		用于存放用户定义的接收结束字符。 CPU 将以该字符作为接收的最后一个字节。收到该字符后，无论其它的结束条件如何 CPU 都会立刻终止接收状态。 如果要使本设置值生效，需要将 SM87.5 置为 1。
SMW90		用于存放用户定义的接收准备时间（范围为 1~60000ms）。 执行 RCV 指令后，经过了该时间值，CPU 将自动进入有效接收状态而不管是否收到起始字符，此后接收到的数据将被认为是有效数据。 如果要使本设置值生效，需要将 SM87.4 置为 1。
SMW92		用于存放用户定义的接收字符超时值（范围为 1~60000ms）。 执行 RCV 指令并进入有效接收状态后，若在此超时时间内没有接收到任何字符，CPU 就会结束接收状态，无论其它的结束条件如何。 如果要使本置值生效，需要将 SM87.2 置为 1。
SMW94		用于存放用户定义的每次接收字符数（1~255）。 CPU 只要接收到了此数量个有效字符，无论其它的结束条件如何，都会马上终止接收状态。本设置值总是有效。 若该值设置为 0，则 RCV 指令将直接退出。

7、SMB136 到 SMB165

SMB136到SMB165用于监测和控制高速计数器 HSC 3 、HSC 4和HSC 5的操作，见下表。

SM	描 述
SM136.0~SM136.4	保留
SM136.5	HSC3当前计数方向位：1 – 增计数
SM136.6	HSC3当前值等于预设值位：1 – 等于
SM136.7	HSC3当前值大于预设值位：1 – 大于
SM137.0~SM137.2	保留
SM137.3	HSC3方向控制位：1 – 增计数
SM137.4	HSC3更新方向：1 – 更新方向
SM137.5	HSC3更新预设值：1 – 向HSC0写新的预设值
SM137.6	HSC3更新当前值：1 = 向HSC0写新的当前值

SM137.7	HSC3 有效位：1 – 有效
SMD138	HSC3新当前值。
SMD142	HSC3新预设值。
SM146.0~SM146.4	保留
SM146.5	HSC4当前计数方向位：1 – 增计数
SM146.6	HSC4当前值等于预设值位：1 – 等于
SM146.7	HSC4当前值大于预设值位：1 – 大于
SM147.0	HSC4复位有效电平控制位：0 – 高电平，1– 低电平
SM147.1	保留
SM147.2	HSC4正交计数器速率选择：0 –4×速率，1 – 1×速率
SM147.3	HSC4方向控制位：1 – 增计数
SM147.4	HSC4更新方向：1 – 更新方向
SM147.5	HSC4更新预设值：1 – 向HSC4写新的预设值
SM147.6	HSC4更新当前值：1 – 向HSC4写新的当前值
SM147.7	HSC4有效位：1 – 有效
SMD148	HSC4新当前值。
SMD152	HSC4新预设值。
SM156.0~SM156.4	保留
SM156.5	HSC5当前计数方向位：1 – 增计数
SM156.6	HSC5当前值等于预设值位：1 – 等于
SM156.7	HSC5当前值大于预设值位：1 – 大于
SM157.0~SM157.2	保留
SM157.3	HSC5方向控制位：1 – 增计数
SM157.4	HSC5更新方向：1 – 更新方向
SM157.5	HSC5更新预设值：1 – 向HSC0写新的预设值
SMD158	HSC5新当前值。
SMD	HSC5新预设值。

8、SMB166 到 SMB194

SMB166~SMB194 用来显示包络步的数量和V区中包络表的地址和中表的地址。

SM	描　　　述
SMB166	PTO 0包络步当前计数值
SMB167	保留
SMW168	PTO 0的包络表在V区中的地址。
SMB170~SMB175	保留
SMB176	PTO 1的包络步当前计数值
SMB177	保留
SMW178	PTO 1的包络表在V区中的地址。
SM180～SM194	保留